Theoretische Informatik

Stefan Hetzl
stefan.hetzl@tuwien.ac.at

TU Wien

SS 2024

Inhaltsverzeichnis

Einführung

Dieses Skriptum bietet eine Einführung in die folgenden drei Gebiete:

1. Automatentheorie
2. Komplexitätstheorie
3. Berechenbarkeitstheorie

Gemeinsam ist diesen drei Gebieten dass sie sich damit beschäftigten, die Komplexität von Teilmengen einer abzählbar unendlichen Menge, z.B. den natürlichen Zahlen, zu messen. In diesem Kontext wird unter "Komplexität" algorithmische Komplexität verstanden, d.h. für ein solchen $X \subseteq \mathbb{N}$: wie schwierig ist es die folgende Frage zu beantworten: gegeben ein $n \in \mathbb{N}$, ist $n \in X$?

Wie schwierig es ist diese Frage zu beantworten kann mit unterschiedlichem X sehr stark variieren. Ist X zum Beispiel die Menge der geraden Zahlen, dann kann, gegeben ein $n \in \mathbb{N}$ in Dezimalnotation durch einfacher Fallunterscheidung auf der Einerstelle sofort beantwortet werden ob $n \in X$ ist. Ist zum Beispiel $X = \mathbb{P}$ dann ist es auch möglich einen Algorithmus anzugeben der die Frage ob $n \in X$ ist beantwortet, allerdings wird dieser offensichtlich etwas komplexer sein müssen.

Diese Gebiete unterscheiden sich in der Komplexität der Mengen die sie betrachten. In obiger Liste sind sie nach aufsteigender Komplexität sortiert, allerdings führt die Betrachtung komplexerer Mengen nicht notwendigerweise auch zu schwierigeren mathematischen Fragestellungen.

Diese drei Gebiete haben unterschiedliche historische Wurzeln: die Automatentheorie ist stark in der Linguistik verwurzelt, wird heutzutage aber hauptsächlich in der Informatik angewandt. Das Thema der Komplexitätstheorie ist "effiziente Berechenbarkeit" in einem weiten Sinn. Dieses Gebiet ist auf recht direktem Weg aus der Informatik entstanden. Berechenbarkeitstheorie ist ein wohl-etabliertes Teilgebiet der mathematischen Logik und deutlich älter als die beiden anderen.

Zur weiterführenden Lektüre können die folgenden Bücher empfohlen werden: [4], [1] vor allem für die Automatentheorie, [3] vor allem für die Komplexitätstheorie, und [2] für die Berechenbarkeitstheorie.

Kapitel 1

Reguläre Sprachen

1.1 Endliche Automaten

Wir beginnen dieses Kapitel mit einigen grundlegenden Begriffen und Operationen. Ein *Alphabet* ist eine endliche Menge von Symbolen. Wir verwenden $A, A', \ldots$ für Alphabete. Als Symbole verwenden wir typischerweise $a, b, c, 0, 1, \ldots$ und als Variablen für Symbole $x, y, z, \ldots$. Ein *A-Wort* ist eine endliche Folge von Symbolen aus A. Oft ist A aus dem Kontext heraus klar, dann sprechen wir einfach über "Wörter" statt über "A-Wörter" Für Wörter verwenden wir üblicherweise $w, v, u, \ldots$. Die leere Folge von Symbolen, das *Leerwort*, wird als ε geschrieben. Die Menge aller A-Wörter wird als A^* geschrieben. Eine *Sprache* ist eine Menge $L \subseteq A^*$.

Definition 1.1. Wir definieren die folgenden Operationen auf Wörtern:

1. *Verkettung*: Für Wörter $v = x_1 \cdots x_n$ und $w = y_1 \cdots y_k$ definieren wir das Wort $v \cdot w$ als $x_1 \cdots x_n y_1 \cdots y_k$. Oft schreiben wir dafür einfach vw.
2. *Potenz*: Für ein Wort w und ein $k \in \mathbb{N}$ definieren wir w^k rekursiv durch $w^0 = \varepsilon$ und $w^{k+1} = ww^k$.
3. *Länge*: Für ein Wort $w = x_1 \cdots x_n$ definieren wir $|w| = n$.
4. Für ein Wort w und einen Buchstaben x schreiben wir $n_x(w)$ für die Anzahl der Vorkommen des Buchstabens x in w.

Algebraisch betrachtet ist $(A^*, \cdot, \varepsilon)$ das von A frei erzeugte Monoid.

Wir definieren nun einige grundlegende Operationen auf Sprachen. Nachdem eine Sprache eine Menge von Wörtern ist haben Mengenoperationen wie die Vereinigung $\cup$, der Durchschnitt $\cap$ oder das Komplement $\cdot^{\mathrm{c}}$ die gewohnte Bedeutung. Man beachte, dass das Komplement relativ zu einem Alphabet als $L^{\mathrm{c}} = A^* \backslash L$ verstanden werden muss.

Definition 1.2. Wir definieren die folgenden Operationen auf Sprachen:

1. *Verkettung*: Für Sprachen L_1 und L_2 definieren wir die Verkettung $L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$. Wie bei Wörtern schreiben wir auch hier oft einfach $L_1 L_2$.
2. *Potenz*: Für eine Sprache L und ein $k \in \mathbb{N}$ definieren wir $L^0 = \{\varepsilon\}$ und $L^{k+1} = L \cdot L^k$.
3. *Kleene-Stern*[1]: Für eine Sprache L definieren wir $L^* = \bigcup_{k \geqslant 0} L^k$.

[1] benannt nach Stephen C. Kleene (1909–1994)

Damit ist es möglich die Notation A^* für die Menge aller Wörter über A als Spezialfall des Kleene-Sterns zu verstehen. Oft schreiben wir auch L^+ für $LL^* = L^*L = \bigcup_{k \geqslant 1} L^k$.

Beispiel 1.3. $\{a, b\}^*$ ist die Menge aller Wörter die nur aus a und b bestehen.

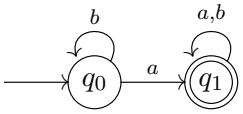
Es ist einfach zu zeigen, dass $(\mathcal{P}(A^*), \cup, \varnothing, \cdot, \{\varepsilon\})$ ein Halbring ist: $(\mathcal{P}(A^*), \cup, \varnothing)$ ist ein kommutatives Monoid, $(\mathcal{P}(A^*), \cdot, \{\varepsilon\})$ ist ein Monoid und es gelten die Distributivgesetze $L_1 \cdot (L_2 \cup L_3) = L_1L_2 \cup L_1L_3$ und $(L_1 \cup L_2) \cdot L_3 = L_1L_3 \cup L_2L_3$. Wie üblich in Halbringen arbeiten wir auch hier mit der Konvention dass die Multiplikation $\cdot$ stärker bindet als die Addition $\cup$.

Beispiel 1.4. Sei $L = (\{c, b\}^*\{a\}^+\{b\})^*\{c, b\}^*$. Um die Notation zu vereinfachen lassen wir oft die Mengenklammern um Singletonmengen weg. Damit können wir dann L schreiben als $(\{c, b\}^*a^+b)^*\{c, b\}^*$. Sei weiters $R = A^*\backslash(A^*acA^* \cup A^*a)$. Wir wollen zeigen dass $L = R$ ist.

Für die Inklusion von links nach rechts sei $w \in L$. Dann existieren $n \geqslant 0, k_1, \ldots, k_n \geqslant 1, v_1, \ldots, v_{n+1} \in \{b, c\}^*$ so dass $w = v_1a^{k_1}bv_2a^{k_2}b \cdots v_na^{k_n}bv_{n+1}$. Damit ist $w \in A^*$. Außerdem ist $w \notin A^*acA^*$ weil ac nicht als Teilwort von w vorkommt. Weiters ist $w \notin A^*a$ weil $w = \varepsilon$ ist oder w mit $bv_{n+1} \in b\{b, c\}^*$ endet.

Für die Inklusion von rechts nach links zeigen wir: $w \in R$ impliziert $w \in L$ mit Induktion nach $n_a(w)$. Falls $n_a(w) = 0$ dann ist $w \in \{b, c\}^*$. Falls $n_a(w) \geqslant 1$ sei $w = w_1a^kw_2$ wobei $k \geqslant 1$, $w_1 \in \{b, c\}^*$ und w_2 nicht mit einem a beginnt. Zunächst ist $w_2 \neq \varepsilon$ weil $w \notin A^*a$. Also ist $w = w_1a^kxw_2'$ mit $x \in \{b, c\}$. Da $w \notin A^*acA^*$ ist $x \neq c$, also $w = w_1a^kbw_2'$ mit $w_2' \in A^*\backslash(A^*acA^* \cup A^*a)$. Nach Induktionshypothese ist $w_2' \in L$ und damit auch $w \in L$.

Beispiel 1.5. Sei $L = b^*a\{a, b\}^*$. Intuitiv ist klar, dass wir die Sprache L durch den folgenden Automaten darstellen können.



Wir präzisieren nun den Begriff des endlichen Automaten.

Definition 1.6. Ein *deterministischer endlicher Automat* (engl. *deterministic finite automaton, DFA*) ist ein Tupel $D = (Q, A, \delta, q_0, F)$ wobei Q eine endliche Menge von *Zuständen* ist, A ein Alphabet, $\delta : Q \times A \to Q$ die *Übergangsfunktion*, $q_0 \in Q$ der *Startzustand* und $F \subseteq Q$ ist die Menge der *Endzustände*.

Definition 1.7. Die Übergangsfunktion δ eines DFA wird durch die folgende rekursive Definition zu $\delta : Q \times A^* \to Q$ erweitert:

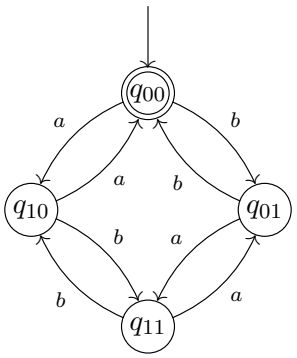
$$\begin{aligned} \delta(q, \varepsilon) &= q \\ \delta(q, xw) &= \delta(\delta(q, x), w) \end{aligned}$$

Definition 1.8. Sei $D = (Q, A, \delta, q_0, F)$ ein DFA. Die *von D akzeptierte Sprache* ist $\mathrm{L}(D) = \{w \in A^* \mid \delta(q_0, w) \in F\}$.

Beispiel 1.9. Die Sprache

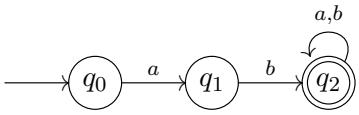
$$L = \{w \in \{a, b\}^* \mid w \text{ enthält eine gerade Anzahl von } a\text{'s und eine gerade Anzahl von } b\text{'s}\}$$

hat den folgenden natürlichen DFA:

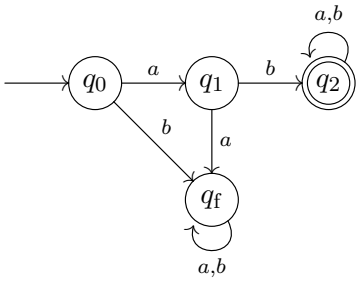


Dieser Automat befindet sich im Zustand q_{ij} genau dann wenn das bisher gelesene Wort i modulo 2 viele a's und j modulo 2 viele b's enthält.

Man beachte, dass die Übergangsfunktion δ eines DFA die Eigenschaft hat, dass sie, für jedes $(q, x) \in Q \times A$ *genau ein* $\delta(q, x) \in Q$ definiert, den nächsten Zustand. Alle bisher betrachteten Diagramme haben ebenfalls diese Eigenschaft und definieren daher jeweils einen eindeutigen DFA. Manchmal ist es allerdings hilfreich, Diagramme zu verwenden die nicht für jedes $(q, x) \in Q \times A$ einen nächsten Zustand spezifizieren. Zum Beispiel suggeriert die Definition der Sprache $L = \{abw \mid w \in \{a, b\}^*\}$ ein Diagramm wie



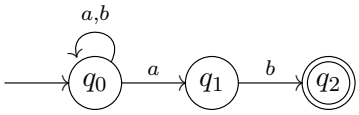
Dieses Diagramm spezifiziert allerdings a priori keinen DFA, da die Werte der Übergangsfunktion auf (q_0, b) und (q_1, a) nicht definiert sind. Wir können allerdings solche Diagramme wie folgt als Spezifikation eines DFA betrachten. Basierend auf der Intuition, dass ein Wort akzeptiert wird genau dann wenn es einen Pfad vom Startzustand in einen Endzustand induziert können wir den Automaten als hängen geblieben betrachten wenn er im Zustand q_0 auf ein b oder im Zustand q_1 auf ein a trifft und dann stipulieren dass das Wort nicht akzeptiert wird. Das kann mit Hilfe unseres formalen Begriffs des DFA präzise gemacht werden indem wir mit solchen Diagrammen einen Automaten spezifizieren, der noch einen zusätzlichen Zustand, eine *Falle*, enthält in den alle undefinierten Übergänge führen und aus dem kein Übergang mehr herausführt. Das obige Diagramm spezifiziert also den selben DFA wie das folgende:



1.2 Nichtdeterminismus

In einem Diagramm zur Spezifikation eines Automaten ist es auch möglich mehr als einen nächsten Zustand für ein Paar $(q, x) \in Q \times A$ anzugeben. So kommen wir auf natürliche Weise zum Begriff des nichtdeterministischen endlichen Automaten.

Beispiel 1.10. Ein natürlicher Automat für die Sprache $L = \{wab \mid w \in \{a,b\}^*\}$ wäre



wobei der Zyklus am Zustand q_0 dazu verwendet wird $w \in \{a,b\}^*$ zu lesen und die Zustände q_1 und q_2 um die Buchstaben a und b am Ende des Worts zu lesen. Dieses Diagramm beschreibt allerdings keinen DFA da der Wert von $\delta(q_0, a)$ nicht eindeutig ist.

Definition 1.11. Ein *nichtdeterministischer endlicher Automat* (engl. *nondeterministic finite automaton, NFA*) ist ein Tupel $N = (Q, A, \Delta, q_0, F)$ wobei Q eine endliche Menge von Zuständen ist, A ein Alphabet, $\Delta \subseteq Q \times A \times Q$ die *Übergangsrelation*, $q_0 \in Q$ der Startzustand und $F \subseteq Q$ die Menge der Endzustände.

Definition 1.12. Die Übergangsrelation Δ eines NFA wird zu $\Delta \subseteq Q \times A^* \times Q$ erweitert durch die Definition:

$$(q, \varepsilon, q) \in \Delta \text{ für alle } q \in Q$$
$$(p, xw, r) \in \Delta \text{ falls ein } q \in Q \text{ existiert so dass } (p, x, q) \in \Delta \text{ und } (q, w, r) \in \Delta$$

Definition 1.13. Sei $N = (Q, A, \Delta, q_0, F)$ ein NFA. Die von N akzeptierte Sprache ist $\mathrm{L}(N) = \{w \in A^* \mid \exists q \in F \text{ so dass } (q_0, w, q) \in \Delta\}$.

Man beachte dass in der obigen Definition die Existenz *eines einzigen* akzeptierenden Pfades für w hinreichend für die Akzeptanz von w ist. Alle anderen Pfade können in Nicht-Endzuständen enden.

Beispiel 1.14. Das in Beispiel 1.10 angegebene Diagramm beschreibt den NFA $N = (Q, A, \Delta, q_0, F)$ wobei $Q = \{q_0, q_1, q_2\}$, $A = \{a, b\}$, $\Delta = \{(q_0, a, q_0), (q_0, b, q_0), (q_0, a, q_1), (q_1, b, q_2)\}$ und $F = \{q_2\}$.

Satz 1.15. *Sei $L \subseteq A^*$. Dann existiert ein DFA D mit $\mathrm{L}(D) = L$ genau dann wenn ein NFA N existiert mit $\mathrm{L}(N) = L$.*

Beweis. Die Richtung von links nach rechts ist trivial: jede Übergangsfunktion kann als Übergangsrelation betrachtet werden. Für die andere Richtung sei $N = (Q, A, \Delta, q_0, F)$ ein NFA. Wir definieren den DFA $D = (\mathcal{P}(Q), A, \delta, \{q_0\}, \{S \subseteq Q \mid S \cap F \neq \varnothing\})$ wobei

$$\delta(S, x) = \{q \in Q \mid \exists p \in S \text{ so dass } (p, x, q) \in \Delta\}.$$

Zunächst zeigen wir, mit Induktion nach $|w|$, dass

$$\delta(S, w) = \{q \in Q \mid \exists p \in S \text{ so dass } (p, w, q) \in \Delta\}. \qquad (*)$$

Für das Leerwort gilt

$$\delta(S, \varepsilon) = S = \{q \in Q \mid \exists p \in S \text{ so dass } (p, \varepsilon, q) \in \Delta\}$$

und für ein beliebiges Wort $w = xv$ gilt

$$\begin{aligned}\delta(S, xv) = \delta(\delta(S, x), v) &=^{\mathrm{IH}} \{q \in Q \mid \exists p \in \delta(S, x) \text{ so dass } (p, v, q) \in \Delta\} \\ &= \{q \in Q \mid \exists p' \in S, p \in Q \text{ so dass } (p', x, p), (p, v, q) \in \Delta\} \\ &= \{q \in Q \mid \exists p' \in S \text{ so dass } (p', xv, q) \in \Delta\}.\end{aligned}$$

Somit erhalten wir

$$\begin{aligned}\mathrm{L}(D) &= \{w \in A^* \mid \delta(\{q_0\}, w) \cap F \neq \varnothing\} \\ &=^{(*)} \{w \in A^* \mid \exists q \in F \text{ so dass } (q_0, w, q) \in \Delta\} \\ &= \mathrm{L}(N).\end{aligned}$$

□

Der obige Beweis wird in der Literatur auch als "Potenzmengenkonstruktion" bezeichnet da wir einen DFA aufbauen dessen Zustände die Teilmengen von Zuständen des gegebenen NFAs sind.

Wenn eine bestimme Klasse von mathematischen Objekten (wie zum Beispiel eine Menge von Sprachen) auf verschiedene Arten beschrieben werden kann (wie zum Beispiel durch DFAs und NFAs) ist das ein Indiz dafür, dass es sich um eine robuste und daher auch wichtige Klasse handelt. Das ist auch hier der Fall und so definieren wir:

Definition 1.16. Ein Sprache $L \subseteq A^*$ heißt *regulär* wenn ein DFA D existiert so dass $\mathrm{L}(D) = L$, bzw. ein NFA N so dass $\mathrm{L}(N) = L$.

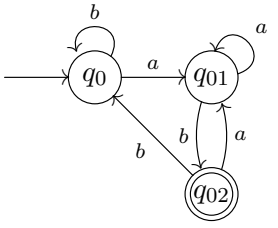
Beispiel 1.17. In Fortsetzung von Beispiel 1.10 wollen wir nun einen DFA finden, der die selbe Sprache akzeptiert. Im Prinzip ist es möglich die Potenzmengenkonstruktion aus dem Beweis von Satz 1.15 wörtlich anzuwenden. In diesem Beispiel würde die Vorgehensweise zu einem DFA mit $2^3 = 8$ Zuständen führen. In der Praxis ist es allerdings geschickter nur jene Zustände zu konstruieren die tatsächlich vom Startzustand $\{q_0\}$ erreichbar sind. Diese Menge von Zuständen kann auf systematische Weise wie folgt bestimmt werden: Wir erstellen eine Tabelle mit dem Startzustand und den Symbolen des Alphabets.

	a	b
$\{q_0\}$		

Der Eintrag in der i-ten Zeile und der j-ten Spalte soll die Menge von Zuständen des NFA enthalten die von einem Zustand aus der Beschriftung der i-ten Zeile mit dem Symbol das die j-Spalte beschriftet erreichbar sind. Falls dieser Prozess einen neuen Zustand des DFA erzeugt, so wird dieser als neue, leere, Zeile der Tabelle hinzugefügt. Diese Tabelle ist saturiert wenn jeder Zustand des DFA der in der Tabelle vorkommt auch als Beschriftung einer Zeile vorkommt. Sobald die Tabelle saturiert ist, ist die Konstruktion des NFA vollständig. Die saturierte Tabelle in diesem Beispiel ist:

	a	b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$

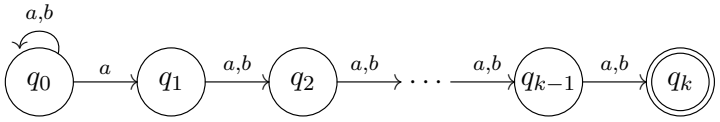
Auf diese Weise haben wir also den folgenden DFA konstruiert, der per constructionem äquivalent zu dem NFA aus Beispiel 1.10 ist:



Wir haben also gesehen, dass NFAs und DFA die selbe Menge von Sprachen beschreiben, d.h. dass sie *extensional äquivalent* sind. Eine Frage, die durch dieses Resultat nicht beantwortet wird, ist ob sie auch gleich kompakte Darstellungen der selben Sprachen erlauben. Es ist klar, dass die Darstellung einer regulären Sprache durch einen NFA höchstens so viele Zustände benötigt wie die Darstellung der selben Sprache durch einen DFA, da jeder DFA trivialerweise als NFA betrachtet werden kann. Für die andere Richtung können wir zunächst nur beobachten, dass die Potenzmengenkonstruktion im schlechtesten Fall exponentiell ist. Gibt es eine andere Konstruktion die eine besser Komplexität hat? Der folgende Satz zeigt, dass die Antwort auf diese Frage nein ist.

Satz 1.18. *Für alle $k \geqslant 1$ gibt es eine Sprache $L_k \subseteq \{a,b\}^*$ und einen NFA N_k mit $k+1$ Zuständen so dass $\mathrm{L}(N_k) = L_k$, aber jeder DFA D mit $L(D) = L_k$ hat mindestens 2^k Zustände.*

Beweis. Sei $L_k = \{wav \mid w, v \in \{a,b\}^*, |v| = k-1\}$. Wir definieren den NFA N_k als:



Sei $D = (Q, A, \delta, q_0, F)$ ein DFA mit $\mathrm{L}(D) = L_k$ und $|Q| < 2^k$. Aus dem Schubfachprinzip folgt dass es Wörter $\overline{x} = x_1 \cdots x_k$ und $\overline{y} = y_1 \cdots y_k$ gibt so dass

$$\delta(q_0, \overline{x}) = \delta(q_0, \overline{y}) \text{ und } \overline{x} \neq \overline{y}.$$

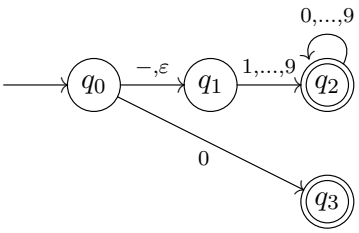
Also gibt es ein i so dass $x_i \neq y_i$. Sei o.B.d.A. $x_i = a$ und $y_i = b$. Wir definieren

$$u = \overline{x}a^{i-1}, \text{ und } v = \overline{y}a^{i-1}.$$

Nun ist der k-te Buchstabe von rechts in u ein a und in v ein b, also ist $u \in L_k$ aber $v \notin L_k$. Trotzdem haben wir $\delta(q_0, \overline{x}w) = \delta(q_0, \overline{y}w)$ für alle $w \in \{a,b\}^*$, insbesondere $\delta(q_0, u) = \delta(q_0, v)$. Widerspruch. □

Gelegentlich ist es praktisch wenn wir in einem endlichen Automaten einen Übergang erlauben ohne einen Buchstaben einzulesen. Eine Klasse von Automaten die das erlaubt, sind NFAs mit ε-Übergängen.

Beispiel 1.19. Mit NFAs mit ε-Übergängen kann zum Beispiel die Sprache der Dezimalnotationen für ganze Zahlen wie folgt akzeptiert werden: Sei $A = \{0, 1, \ldots, 9, -\}$ und definiere einen NFA mit ε-Übergängen als:



Dieser Automat verwendet einen ε-Übergang für die Behandlung des optionalen Vorzeichens $-$.

Definition 1.20. Ein *nichtdeterministischer endlicher Automat mit ε-Übergängen (ε-NFA)* ist definiert wie ein NFA als (Q, A, Δ, q_0, F) wobei jetzt $\Delta \subseteq Q \times (A \cup \{\varepsilon\}) \times Q$.

Die erweiterte Übergangsrelation $\Delta \subseteq Q \times A^* \times Q$ sowie die Sprache $\mathrm{L}(N)$ eines ε-NFAs ist analog zu den entsprechenden Begriffen eines NFAs definiert. Dann gilt auch:

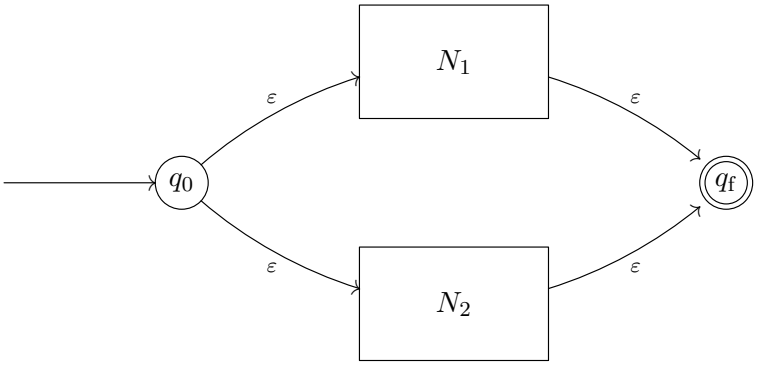
Satz 1.21. *Eine Sprache $L \subseteq A^*$ ist regulär genau dann wenn es einen ε-NFA N gibt mit* $\mathrm{L}(N) = L$.

Beweisskizze. Im Wesentlichen kann man hier vorgehen wie im Beweis von Satz 1.15. Die Potenzmengenkonstruktion wird erweitert um die Betrachtung der ε-Hülle eines Zustands q, d.h. der Menge aller Zustände die von q aus durch ε-Übergänge erreichbar sind. Eine detaillierte Darstellung dieses Beweises ist z.B. in [1, Theorem 2.2] zu finden. □

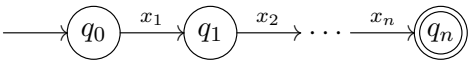
Satz 1.22. *Die Klasse der regulären Sprachen hat die folgenden Abschlusseigenschaften:*

1. *Falls L_1 und L_2 regulär sind, dann ist auch $L_1 \cup L_2$ regulär.*
2. *Falls L regulär ist, dann ist auch L^{c} regulär.*
3. *Falls L_1 und L_2 regulär sind, dann ist auch $L_1 \cap L_2$ regulär.*
4. *Jede endliche Sprache ist regulär.*
5. *Falls L_1 und L_2 regulär sind, dann ist auch $L_1 \cdot L_2$ regulär.*
6. *Falls L regulär ist, dann ist auch L^* regulär.*

Beweis. 1. Seien N_1 und N_2 ε-NFAs für L_1 und L_2 wie oben. Wir erhalten einen ε-NFA für $L_1 \cup L_2$ durch:

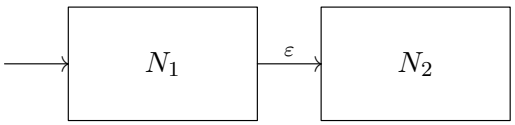


2. Sei $D = (Q, A, \delta, q_0, F)$ ein DFA für L. Dann ist $D' = (Q, A, \delta, q_0, Q \backslash F)$ ein DFA für L^{c}.
3. $L_1 \cap L_2 = (L_1^{\mathrm{c}} \cup L_2^{\mathrm{c}})^{\mathrm{c}}$.
4. Sei $w = x_1 \cdots x_n$ ein Wort mit $x_i \in A$. Dann wird die Sprache $\{w\}$ durch den folgenden Automaten akzeptiert:



   Damit folgt aus 1. dass jede nicht-leere endliche Sprache regulär ist. Aber auch $\varnothing$ ist regulär wie ein beliebiger DFA mit $F = \varnothing$ zeigt.

5. Seien N_1 und N_2 ε-NFAs für L_1 und L_2 wie oben. Dann erhalten wir einen ε-NFA für $L_1 \cdot L_2$ durch



wobei diese Skizze ausdrücken soll, dass jeder Endzustand von N_1 mit dem Startzustand von N_2 durch einen ε-Übergang verbunden wird und die Endzustände von N_1 im neuen Automaten keine Endzustände mehr sind.

6. Sei $N = (Q, A, \Delta, q_0, F)$ ein ε-NFA für L. Dann definieren wir $N' = (Q \cup \{q_0'\}, A, \Delta', q_0', \{q_0'\})$ wobei $\Delta' = \Delta \cup \{(q_0', \varepsilon, q_0)\} \cup \{(q_\mathrm{f}, \varepsilon, q_0') \mid q_\mathrm{f} \in F\}$.

□

1.3 Reguläre Ausdrücke

Eine weitere Charakterisierung der regulären Sprachen kann durch reguläre Ausdrücke gegeben werden. Deren Syntax basiert auf den bereits bekannten Operationen auf Sprachen: Vereinigung, Verkettung und Kleene-Stern. Damit haben sie eine enge Verbindung zur Algebra. Reguläre Ausdrücke (und diverse Erweiterungen) sind ein Standardwerkzeug der Informatik zur Beschreibung von Mustern in Texten, z.B. zur Suche.

Definition 1.23. Sei A ein Alphabet. Die *regulären Ausdrücke über A* sind induktiv wie folgt definiert:

1. Falls $w \in A^*$ dann ist $\{w\}$ ein regulärer Ausdruck über A.
2. $\varnothing$ ist ein regulärer Ausdruck über A.
3. Falls E_1 und E_2 reguläre Ausdrücke über A sind, dann sind auch $E_1 \cup E_2$ und $E_1 \cdot E_2$ reguläre Ausdrücke über A.
4. Falls E ein regulärer Ausdruck über A ist, dann ist auch E^* ein regulärer Ausdruck über A.

Jeder reguläre Ausdruck E definiert auf offensichtliche Weise eine Sprache $\mathrm{L}(E) \subseteq A^*$. Tatsächlich gilt sogar:

Satz 1.24. *Eine Sprache $L \subseteq A^*$ ist regulär genau dann wenn ein regulärer Ausdruck E existiert mit $\mathrm{L}(E) = L$*

Beweis. Für gegebenen E folgt die Regularität von $\mathrm{L}(E)$ mit Induktion nach der Struktur von E aus den in Satz 1.22 bewiesenen Abschlusseigenschaften.

Für die Gegenrichtung sei L regulär und $D = (\{q_1, \ldots, q_n\}, A, \delta, q_1, F)$ ein DFA mit $\mathrm{L}(D) = L$. Für $i, j \in \{1, \ldots, n\}$ und $k \in \{0, \ldots, n\}$ definieren wir[2]

$$L_{i,j}^k = \{w \in A^* \mid \delta(q_i, w) = q_j \text{ und für jedes nicht-leere echte Präfix } v \text{ von } w \text{ gilt: } \delta(q_i, v) = q_l \text{ impliziert } l \leqslant k\}$$

Die Mengen $L_{i,j}^k$ können rekursiv wie folgt definiert werden:

$$L_{i,j}^0 = \begin{cases} \{x \in A \mid \delta(q_i, x) = q_j\} & \text{falls } i \neq j \\ \{x \in A \mid \delta(q_i, x) = q_i\} \cup \{\varepsilon\} & \text{falls } i = j \end{cases}$$
$$L_{i,j}^k = L_{i,k}^{k-1}(L_{k,k}^{k-1})^* L_{k,j}^{k-1} \cup L_{i,j}^{k-1}$$

Nun folgt mit Induktion nach k dass es für jedes $L_{i,j}^k$ einen regulären Ausdruck gibt. Weiters ist $L = \mathrm{L}(D) = \bigcup_{q_j \in F} L_{1,j}^k$, also gibt es auch für L einen regulären Ausdruck. □

[2] $v \in A^*$ heißt *Präfix* von $w \in A^*$ falls ein $u \in A^*$ existiert mit $w = vu$.

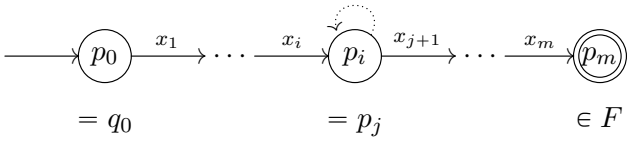
Beispiel 1.25. Die Sprache $L = \{a^i b^j \mid i, j \geqslant 0\}$ ist regulär, da $L = \{a\}^* \cdot \{b\}^*$.

Auf der anderen Seite werden wird bald sehen dass die Sprache $L = \{a^i b^i \mid i \geqslant 0\}$ nicht regulär ist. Um das zu zeigen benötigen wir allerdings das folgende Lemma.

Lemma 1.26 (Pumping Lemma)**.** *Sei L regulär. Dann gibt es ein $n \in \mathbb{N}$ so dass jedes $w \in L$ mit $|w| \geqslant n$ als $w = v_1 v_2 v_3$ geschrieben werden kann wobei*

1. *$v_2 \neq \varepsilon$,*
2. *$|v_1 v_2| \leqslant n$, und*
3. *für alle $k \geqslant 0$: $v_1 v_2^k v_3 \in L$.*

Beweis. Sei $D = (Q, A, \delta, q_0, F)$ ein DFA mit $L(D) = L$. Sei $n = |Q|$, sei $w \in L$ mit $|w| = m \geqslant n$ und $w = x_1 \cdots x_m$ für $x_i \in A$. Der von w in D induzierte Pfad ist: $p_i = \delta(q_0, x_1 \cdots x_i)$ für $i = 0, \ldots, m$. Nachdem D nur n Zustände hat folgt aus dem Schubfachprinzip dass es $i, j \in \{0, \ldots, n\}$ gibt so dass $i < j$ und $p_i = p_j$. Als Diagramm hat der durch w induzierte Pfad die folgende Form:



Wir definieren $v_1 = x_1 \cdots x_i$, $v_2 = x_{i+1} \cdots x_j$ und $v_3 = x_{j+1} \cdots x_m$. Damit erhalten wir 1. aus $i < j$, 2. aus $j \leqslant n$, und 3. aus $p_i = p_j$. □

Dieses Lemma ist ein sehr nützliches Werkzeug um zu zeigen, dass eine gegebene Sprache nicht regulär ist. Sein Name, "Pumping Lemma", auf Deutsch auch "Schleifensatz", kommt daher dass der mittlere Teil v_2 des Wortes w beliebig "aufgepumpt" werden kann, indem die Schleife k mal durchlaufen wird.

Beispiel 1.27. Wir können nun das Pumping Lemma verwenden um zu zeigen dass $L = \{a^i b^i \mid i \geqslant 0\}$ nicht regulär ist. Sei n wie im Pumping Lemma für L und betrachte $w = a^n b^n$. Dann existieren v_1, v_2, v_3 so dass $w = v_1 v_2 v_3$ und die Eigenschaften 1.-3. erfüllt sind. Nachdem $|v_1 v_2| \leqslant n$ ist $v_1 = a^k$ und $v_2 = a^l$. Dann ist also auch $v_1 v_2^2 v_3 = a^{n+l} b^n \in L$. Nachdem aber $v_2 \neq \varepsilon$ ist $l > 0$ und damit $n + l \neq n$, d.h. $a^{n+l} b^n \notin L$. Widerspruch.

Kapitel 2

Kontextfreie Sprachen

2.1 Formale Grammatiken

Eine natürliche Sprache wie Deutsch, Englisch, ... gehorcht einer Grammatik, d.h. einer Sammlung von Regeln welche die Menge der wohlgeformten Sätze definiert. In diesem Abschnitt werden wir sehen wie dieser Begriff mathematisch präzise gemacht werden kann. Auch wenn die Linguistik ein für die Entwicklung der Automatentheorie wichtiges Gebiet war und ist, werden wir uns hier nur mit Grammatiken beschäftigen die zu einfach sind, um die Regeln natürlicher Sprachen zu beschreiben. Diese Grammatiken sind aber ausdrucksstark genug um Programmiersprachen zu beschreiben. Dementsprechend hat die Automatentheorie zahlreiche Anwendungen in der Informatik, z.B. in der Compilerkonstruktion, der Theorie von Programmiersprachen, etc.

Beispiel 2.1. Die Menge der wohlgeformten arithmetischen Ausdrücke über den Variablen x, y, z, wie z.B. $((x + y) \cdot x)$, kann durch die folgenden Regeln definiert werden:

$$A \to V \mid (A + A) \mid (A \cdot A)$$
$$V \to x \mid y \mid z$$

Zur Präzisierung dieser Idee definieren wir:

Definition 2.2. Eine *kontextfreie Grammatik (CFG)* ist ein Tupel (N, T, P, S) wobei N eine endliche Menge von *Nichtterminalsymbolen* ist, T ist eine endliche Menge von *Terminalsymbolen* so dass $T \cap N = \varnothing$, $P \subseteq N \times (N \cup T)^*$ ist eine Menge von *Produktionsregeln* und $S \in N$ ist das *Startsymbol*.

Die Rolle der Menge der Terminalsymbole T ist analog zu der des Alphabets A in der Spezifikation eines Automaten. Statt (N, w) schreiben wir $N \to w$ für eine Produktionsregel. Diese Notation wird mit $N \to w_1 \mid \cdots \mid w_n$ als Abkürzung der Menge $\{(N, w_i) \mid 1 \leqslant i \leqslant n\}$ erweitert.

Beispiel 2.3. Die in Beispiel 2.1 angegebenen Regeln können nun als Produktionsregeln einer kontextfreien Grammatik mit $N = \{A, V\}$, $T = \{x, y, z, +, \cdot, (,)\}$ und Startsymbol A interpretiert werden.

Definition 2.4. Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik. Für alle $w, w' \in (N \cup T)^*$ definieren wir die *1-Schritt Ableitungsrelation* als: $w \Longrightarrow_G w'$ genau dann wenn $w = w_1 A w_2$, $w' = w_1 v w_2$ und $A \to v \in P$. Weiters definieren wir die *Ableitungsrelation* $\Longrightarrow_G^*$ als reflexive und transitive Hülle von $\Longrightarrow_G$.

Die von G erzeugte Sprache ist $\mathrm{L}(G) = \{w \in T^* \mid S \Longrightarrow_G^* w\}$.

Wenn die Grammatik G aus dem Kontext heraus klar ist, dann schreiben wir auch einfach $\Longrightarrow$ und $\Longrightarrow^*$.

Beispiel 2.5. Eine Ableitung in der in Beispiel 2.3 angegebenen kontextfreien Grammatik ist

$$\begin{aligned} A &\Longrightarrow (A \cdot A) \Longrightarrow ((A + A) \cdot A) \Longrightarrow ((V + A) \cdot A) \Longrightarrow ((x + A) \cdot A) \\ &\Longrightarrow ((x + V) \cdot A) \Longrightarrow ((x + y) \cdot A) \Longrightarrow ((x + y) \cdot V) \Longrightarrow ((x + y) \cdot x). \end{aligned}$$

Definition 2.6. Eine Sprache $L \subseteq A^*$ heißt *kontextfrei* falls eine CFG G existiert mit $\mathrm{L}(G) = L$.

Definition 2.7. Eine kontextfreie Grammatik $G = (N, T, P, S)$ heißt *strikt rechtslinear* falls jede Produktion von G eine der folgenden Formen hat:

1. $A \to x$ für $x \in T$,
2. $A \to xB$ für $x \in T$ und $B \in N$ oder
3. $A \to \varepsilon$.

Der Name strikt rechtslinear erklärt sich daher dass Grammatiken die Produktionen der Formen $A \to w$, $A \to wB$ und $A \to \varepsilon$ für $w \in T^*$ enthalten als *rechtslinear* bezeichnet werden. Die Bezeichnung linear verweist auf die Tatsache dass jede Produktion höchstens ein Nichterminalsymbol (eine Variable) enthält.

Beispiel 2.8. Sei $N = \{A, B\}$, $T = \{a, b\}$, $P =$

$$\begin{aligned} A &\to aA \mid aB, \\ B &\to bB \mid b, \end{aligned}$$

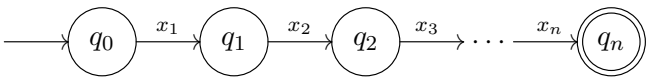
und $G = (N, T, P, A)$. Dann ist G strikt rechtslinear und $\mathrm{L}(G) = \{a^i b^j \mid i, j \geqslant 1\}$.

Satz 2.9. *Eine Sprache $L \subseteq A^*$ ist regulär genau dann wenn eine strikt rechtslineare Grammatik G existiert mit* $\mathrm{L}(G) = L$.

Beweis. Von links nach rechts sei $N = (Q, A, \Delta, q_0, F)$ ein NFA für L. Definiere die Grammatik $G = (N, T, P, S)$ durch $N = Q$, $T = A$,

$$P = \{q \to ap \mid (q, a, p) \in \Delta\} \cup \{q \to \varepsilon \mid q \in F\},$$

und $S = q_0$. Dann behaupten wir, dass $\mathrm{L}(G) = \mathrm{L}(N)$. Sei dazu $w = x_1 \cdots x_n \in A^*$. Dann existiert ein Pfad der Form



in N genau dann wenn auch eine Ableitung der Form

$$q_0 \Longrightarrow x_1 q_1 \Longrightarrow x_1 x_2 q_2 \Longrightarrow \cdots \Longrightarrow x_1 \cdots x_n q_n \Longrightarrow x_1 \cdots x_n$$

in G existiert. Also ist $w \in \mathrm{L}(N)$ genau dann wenn $w \in \mathrm{L}(G)$.

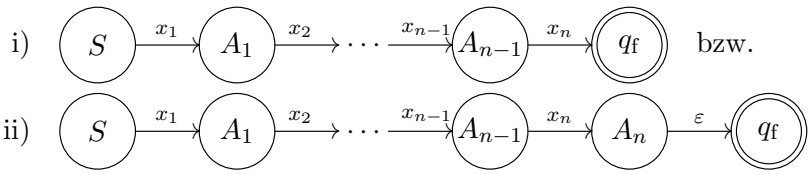
Von rechts nach links sei $G = (N, T, P, S)$ eine strikt rechtslineare Grammatik. Wir definieren einen ε-NFA $M = (Q, A, \Delta, S, F)$ durch $Q = N \cup \{q_\mathrm{f}\}$, $A = T$,

$$\Delta = \{(B, x, C) \mid B \to xC \in P\} \cup \{(B, x, q_\mathrm{f}) \mid B \to x \in P\} \cup \{(B, \varepsilon, q_\mathrm{f}) \mid B \to \varepsilon \in P\},$$

und $F = \{q_\mathrm{f}\}$. Dann behaupten wir wiederum dass $\mathrm{L}(M) = \mathrm{L}(G)$. Sei dazu $w = x_1 \cdots x_n \in T^*$. Dann existiert eine Ableitung der Form

i) $S \Longrightarrow x_1 A_1 \Longrightarrow x_1 x_2 A_2 \Longrightarrow \cdots \Longrightarrow x_1 \cdots x_{n-1} A_n \Longrightarrow x_1 \cdots x_n$ bzw.
ii) $S \Longrightarrow x_1 A_1 \Longrightarrow x_1 x_2 A_2 \Longrightarrow \cdots \Longrightarrow x_1 \cdots x_{n-1} A_n \Longrightarrow x_1 \cdots x_n A_n \Longrightarrow x_1 \cdots x_n$

genau dann wenn auch ein Pfad der Form



in G existiert. Also ist $w \in \mathrm{L}(G)$ genau dann wenn $w \in \mathrm{L}(N)$.

□

Am obigen Beweis sieht man, dass die Nichtterminale einer strikt rechtslinearen Grammatik den Zuständen eines nicht-deterministischen Automaten entsprechen. Das obige Resultat zeigt direkt dass jede reguläre Sprache kontextfrei ist. Die Gegenrichtung ist nicht wahr.

Beispiel 2.10. In Beispiel 1.27 haben wir gezeigt, dass $L = \{a^k b^k \mid k \geqslant 0\}$ nicht regulär ist. Auf der anderen Seite kann L aber durch die kontextfreie Grammatik

$$S \to aSb \mid \varepsilon$$

erzeugt werden, L ist also kontextfrei.

Satz 2.11. *Die Klasse der kontextfreien Sprachen hat die folgenden Abschlusseigenschaften:*

1. *Jede reguläre Sprache ist kontextfrei.*
2. *Falls L_1 und L_2 kontextfrei sind, dann ist auch $L_1 \cup L_2$ kontextfrei.*
3. *Falls L_1 und L_2 kontextfrei sind, dann ist auch $L_1 \cdot L_2$ kontextfrei.*
4. *Falls L kontextfrei ist, dann ist auch L^* kontextfrei.*

Beweis. 1. folgt direkt aus Satz 2.9.

Für 2. seien $G_1 = (N_1, T_1, P_1, S_1)$ und $G_2 = (N_2, T_2, P_2, S_2)$ kontextfreie Grammatiken für L_1 bzw. L_2. O.B.d.A. nehmen wir an dass $N_1 \cap N_2 = \varnothing$. Dann erzeugt

$$G = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}, S)$$

die Sprache $L_1 \cup L_2$.

Für 3. seien $G_1 = (N_1, T_1, P_1, S_1)$ und $G_2 = (N_2, T_2, P_2, S_2)$ kontextfreie Grammatiken für L_1 bzw. L_2. O.B.d.A. nehmen wir wieder an dass $N_1 \cap N_2 = \varnothing$. Dann erzeugt

$$G = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$$

die Sprache $L_1 \cdot L_2$.

Für 4. sei $G = (N, T, P, S)$ eine kontextfreie Grammatik, die L erzeugt. Dann erzeugt

$$G' = (N \cup \{S'\}, T, P \cup \{S' \to SS' \mid \varepsilon\}, S')$$

die Sprache L^*. □

Es gibt auch eine Darstellung der kontextfreien Sprachen durch Automaten. Die entsprechenden Automaten heißen *Kellerautomaten* (engl. *pushdown automata*). Ein Kellerautomat hat, zusätzlich zu einer endlichen Menge von Zuständen, auch noch einen Stapel (engl. *stack*) mit einem endlichen Alphabet von Stapelsymbolen als Speicher zur Verfügung. Ein Stapel erlaubt nur zwei Operationen: 1. ein neues Element auf die Stapel legen, 2. das oberste Element vom Stapel nehmen. Damit bietet ein Stapel zwar einen unbeschränkten Speicher, allerdings auch einen auf den der Zugriff stark eingeschränkt ist. Wir behandeln Kellerautomaten in dieser Vorlesung nicht, Details können zum Beispiel in [1] oder [4] nachgelesen werden.

2.2 Ableitungsbäume

Auch für kontextfreie Sprachen gibt es einen Schleifensatz (engl. *pumping lemma*). Nachdem es sich dabei um eine größere Klasse von Sprachen handelt, hat auch der Schleifensatz eine komplexere Struktur. Um ihn zu beweisen führen wir zunächst den Begriff des Ableitungsbaums ein und betrachten danach Normalformen für Grammatiken.

Beispiel 2.12. Sei $G = (N, \{x, y, z, \cdot, +, (,)\}, P, A)$ die Grammatik aus Beispiel 2.3. In Beispiel 2.5 haben wir eine Ableitung $A \Longrightarrow_G^* ((x+y)\cdot x)$ angegeben. Dieses Wort hat noch weitere Ableitungen, z.B.

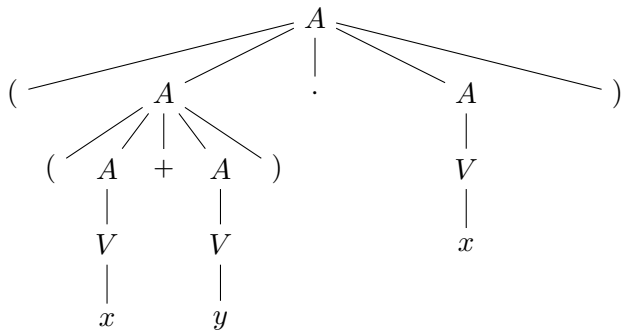
$$\begin{aligned} A &\Longrightarrow (A \cdot A) \Longrightarrow (A \cdot V) \Longrightarrow (A \cdot x) \Longrightarrow ((A + A) \cdot x) \\ &\Longrightarrow ((A + V) \cdot x) \Longrightarrow ((A + y) \cdot x) \Longrightarrow ((V + y) \cdot x) \Longrightarrow ((x + y) \cdot x). \end{aligned}$$

Definition 2.13. Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik, dann heißt

$$S \Longrightarrow_G \alpha_1 \Longrightarrow_G \alpha_2 \Longrightarrow_G \cdots \Longrightarrow_G \alpha_n$$

Linksableitung wenn in jedem Schritt $\alpha_i \Longrightarrow_G \alpha_{i+1}$ das in α_i am weitesten links stehende Nichtterminalsymbol ersetzt wird und *Rechtsableitung* wenn in jedem Schritt $\alpha_i \Longrightarrow_G \alpha_{i+1}$ das in α_i am weitesten rechts stehende Nichtterminalsymbol ersetzt wird.

Beispiel 2.14. Die in Beispiel 2.5 angegebene Ableitung ist eine Linksableitung, die in Beispiel 2.12 angegebene Ableitung ist eine Rechtsableitung. Wir sehen also dass das Wort $((x + y) \cdot x)$ in dieser Grammatik verschiedene Ableitungen hat. Allerdings induzieren beide Ableitungen den folgenden Baum:



Wir präzisieren:

Definition 2.15. Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik. Ein Ableitungsbaum für G ist ein Baum in dem jeder Knoten mit einem Symbol aus $N \cup T \cup \{\varepsilon\}$ beschriftet ist und der die folgenden Bedingungen erfüllt:

1. Jeder innere Knoten ist mit einem $A \in N$ beschriftet.
2. Falls ein Knoten v mit A beschriftet ist und seine Kinder $v_1, \ldots, v_n$ (von links nach rechts nummeriert) mit $\alpha_1, \ldots, \alpha_n \in T \cup N$ beschriftet sind, dann enthält P die Produktion $A \to \alpha_1 \cdots \alpha_n$.
3. Der Baum hat mindestens zwei Knoten.
4. Falls ein Knoten v mit ε beschriftet ist, dann ist v das einzige Kind seines Vaters.

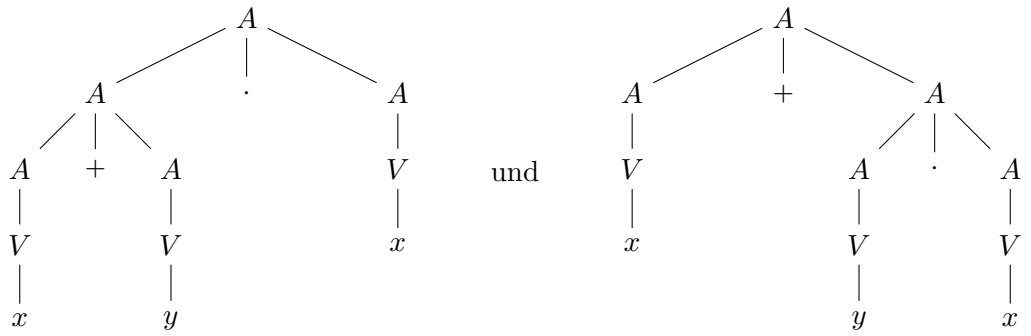
Ein *vollständiger Ableitungsbaum* ist ein Ableitungsbaum dessen Wurzel mit S beschriftet ist und dessen Blätter nur mit $x \in T$ und ε beschriftet sind. Für eine Grammatik G schreiben wir $\mathcal{B}(G)$ für die Menge aller G-Ableitungsbäume sowie $\mathcal{B}_w^A(G)$ für die Menge aller G-Ableitungsbäume des Wortes w deren Wurzel mit A beschriftet ist.

Definition 2.16. Eine kontextfreie Grammatik G heißt *mehrdeutig* falls ein $w \in \mathrm{L}(G)$ existiert so dass zwei verschiedene vollständige G-Ableitungsbäume für w existieren. Ist das nicht der Fall heißt G *eindeutig*.

Beispiel 2.17. Sei $G = (\{A, V\}, \{x, y, z, +, \cdot, (,)\}, P, A)$ wobei $P =$

$$A \to A + A \mid A \cdot A \mid V$$
$$V \to x \mid y \mid z$$

Dann ist $x + y \cdot x \in \mathrm{L}(G)$ und hat die beiden Ableitungsbäume



Die Grammatik G ist also mehrdeutig.

Definition 2.18. Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik. $A \in N$ heißt *nützlich* falls es $u, v \in T^*$ gibt so dass $S \Longrightarrow_G^* uAv$ und $w \in T^*$ so dass $A \Longrightarrow_G^* w$. Eine Grammatik in der alle $A \in N \backslash \{S\}$ nützlich sind heißt *gekürzt*.

Man beachte, dass $\mathrm{L}(G) \neq \varnothing$ genau dann wenn S nützlich ist. Weiters ist $A \in N \backslash \{S\}$ nützlich genau dann wenn ein $w \in T^*$ und ein $B \in \mathcal{B}_w^S(G)$ exisitiert so dass A in B vorkommt.

Lemma 2.19. *Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik, sei $N' = \{S\} \cup \{A \in N \backslash \{S\} \mid$ A ist nützlich in $G\}$, sei $P' = \{A \to \alpha \in P \mid A \in N', \alpha \in (T \cup N')^*\}$ und sei $G' = (N', T, P', S)$. Dann ist G' eine gekürzte kontextfreie Grammatik mit $\mathrm{L}(G') = \mathrm{L}(G)$ und falls G eindeutig ist dann ist das auch G'.*

Beweis. Sei $B \in \mathcal{B}_w^S(G)$. Dann enthält B nur Nichtterminalsymbole die in G nützlich sind, also ist $B \in \mathcal{B}_w^S(G')$. Sei umgekehrt $B' \in \mathcal{B}_w^S(G')$. Dann ist $B' \in \mathcal{B}_w^S(G)$ da ja $N' \subseteq N$ und $P' \subseteq P$. Also ist $\mathcal{B}_w^S(G) = \mathcal{B}_w^S(G')$ was $\mathrm{L}(G) = \mathrm{L}(G')$ und die Beibehaltung der Eindeutigkeit zeigt. Weiters ist G' gekürzt da A in G' nützlich ist genau dann wenn A in einem $B \in \mathcal{B}_w^S(G') = \mathcal{B}_w^S(G)$ vorkommt genau dann wenn A in G nützlich ist. □

Das im obigen Beweis verwendete Argument über die Menge der Ableitungsbäume kann wie im folgenden Lemma etwas verallgemeinert werden.

Lemma 2.20. *Seien G und G' kontextfreie Grammatiken mit Startsymbolen S und S' und den selben Terminalsymbolen T, sei $f : \mathcal{B}(G) \to \mathcal{B}(G')$ so dass $f(\mathcal{B}_w^S(G)) = \mathcal{B}_w^{S'}(G')$ für alle $w \in T^*$. Dann ist $\mathrm{L}(G') = \mathrm{L}(G)$ und falls G eindeutig ist dann ist das auch G'.*

Beweis. Sei $w \in \mathrm{L}(G)$, sei $B \in \mathcal{B}_w^S(G)$, dann ist $f(B) \in \mathcal{B}_w^S(G')$ und damit ist $w \in \mathrm{L}(G')$. Umgekehrt sei $w \in \mathrm{L}(G')$ und sei $B' \in \mathcal{B}_w^S(G')$, dann existiert ein $B \in \mathcal{B}_w^S(G)$ mit $f(B) = B'$ und damit ist $w \in \mathrm{L}(G)$. Angenommen G' wäre mehrdeutig, dann gäbe es ein $w \in \mathrm{L}(G')$ sowie $B_1', B_2' \in \mathcal{B}_w^S(G')$ mit $B_1' \neq B_2'$. Damit gäbe es $B_1, B_2 \in \mathcal{B}_w^S(G)$ mit $f(B_1) = B_1'$ und $f(B_2) = B_2'$ und $B_1 \neq B_2$, also wäre auch G mehrdeutig. □

Definition 2.21. Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik. Eine Produktion der Form $A \to \varepsilon$ für $A \in N$ heißt *ε-Produktion*. Wir sagen dass G *fast keine ε-Produktionen* hat falls $A \to \varepsilon \in P$ impliziert dass $A = S$.

Lemma 2.22. *Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik. Dann existiert eine gekürzte kontextfreie Grammatik $G' = (N', T, P', S')$ mit $\mathrm{L}(G') = \mathrm{L}(G)$ die fast keine ε-Produktionen hat und in der S' nicht auf der rechten Seite von P' vorkommt. Falls G eindeutig ist, dann ist auch G' eindeutig.*

Beweis. Sei $E = \{A \in N \mid A \Longrightarrow_G^* \varepsilon\}$ die Menge der annulierbaren Nichterminalsymbole von G. Sei

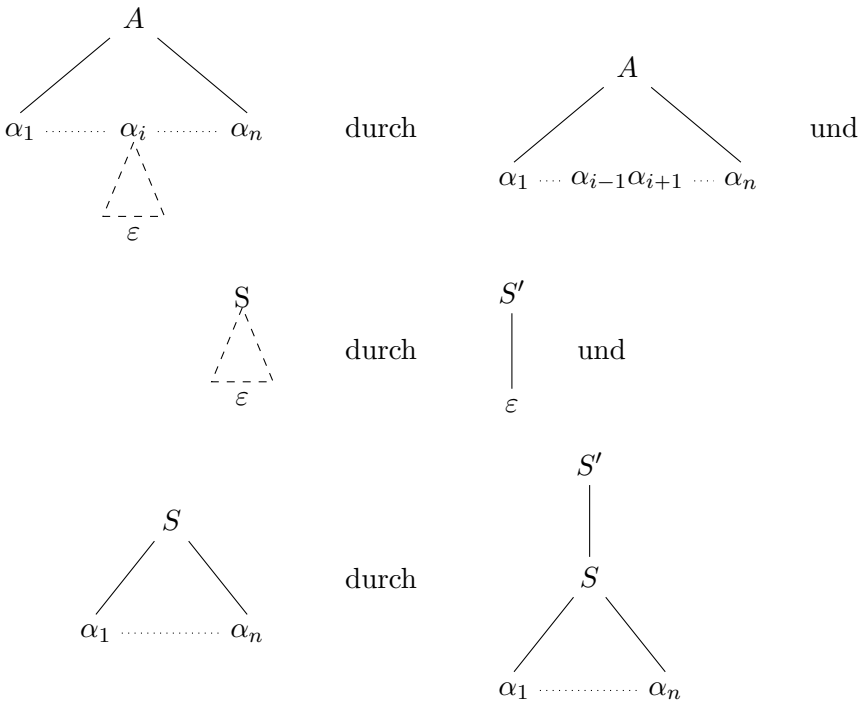
$$P_0 = \{A \to \beta_1 \cdots \beta_n \mid A \to \alpha_1 \cdots \alpha_n \in P, \alpha_1, \ldots, \alpha_n \in T \cup N, \\ \beta_i = \begin{cases} \alpha_i \text{ oder } \varepsilon & \text{falls } \alpha_i \in E \\ \alpha_i & \text{falls } \alpha_i \notin E \end{cases}, \beta_1 \cdots \beta_n \neq \varepsilon\}.$$

Dann enthält P_0 keine ε-Produktionen. Sei weiters

$$P_0' = \begin{cases} P_0 \cup \{S' \to \varepsilon, S' \to S\} & \text{falls } S \in E \\ P_0 \cup \{S' \to S\} & \text{falls } S \notin E \end{cases}$$

und $G_0' = (N \cup \{S'\}, T, P_0', S')$. Wir definieren die Abbildung $f : \mathcal{B}(G) \to \mathcal{B}(G')$ durch Ersetzung

von Teilbäumen der Form



wobei im letzten Fall nicht alle α_i annulliert werden. Dann ist $f(\mathcal{B}_w^S(G)) = \mathcal{B}_w^{S'}(G'_0)$ für alle $w \in T^*$. Eine Anwendung von Lemma 2.20 zeigt dass $\mathrm{L}(G'_0) = \mathrm{L}(G)$ und falls G eindeutig ist, dann ist auch G'_0 eindeutig. Kürzen von G'_0 mit Lemma 2.19 liefert eine Grammatik G' mit den gewünschten Eigenschaften. □

Definition 2.23. Eine Produktion der Form $A \to B$ wobei A und B Nichterminalsymbole sind heißt *Umbenennung.*

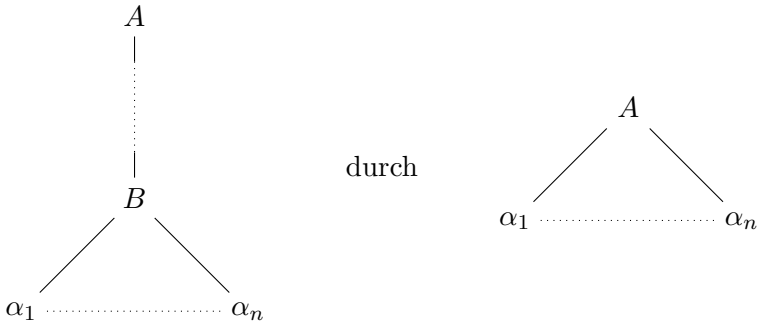
Lemma 2.24. *Sei G eine kontextfreie Grammatik. Dann existiert eine gekürzte kontextfreie Grammatik G' mit $\mathrm{L}(G') = \mathrm{L}(G)$ die fast keine ε-Produktionen und keine Umbenennungen enthält und deren Startsymbol nicht auf der rechten Seite einer Produktion vorkommt. Falls G eindeutig ist, dann ist auch G' eindeutig.*

Beweis. Wir können wegen Lemma 2.22 annehmen, dass $G = (N, T, P, S)$ gekürzt ist, fast keine ε-Produktionen enthält und dass S nicht auf der rechten Seite von P vorkommt. Wir definieren $G_0 = (N, T, P_0, S)$ wobei

$$P_0 = \{A \to \alpha \in P \mid \alpha \notin N\} \cup \{A \to \alpha \mid A \Longrightarrow_G^* B, B \to \alpha \in P, \alpha \notin N\}.$$

Dann enthält G_0 fast keine ε-Produktionen und keine Umbenennungen. Sei $f : \mathcal{B}(G) \to \mathcal{B}(G')$

definiert durch Ersetzung von



wobei $B \to \alpha_1 \cdots \alpha_n$ keine Umbenennung ist. Nun ist $f(\mathcal{B}_w^S(G)) = \mathcal{B}_w^S(G')$ für alle $w \in T^*$ und somit können wir aus Lemma 2.20 folgern dass $\mathrm{L}(G) = \mathrm{L}(G_0)$ und falls G eindeutig ist, dann ist das auch G_0. Aus G_0 erhalten wir durch Kürzen wie in Lemma 2.19 die Grammatik $G' = (N', T, P', S)$. Da $P' \subseteq P_0$ enthält G' fast keine ε-Produktionen und keine Umbenennungen und S' kommt nicht auf der rechten Seite von P' vor. □

Definition 2.25. Eine kontextfreie Grammatik $G = (N, T, P, S)$ ist in *Chomsky Normalform*[1] falls jede Produktion von G eine der folgenden Formen hat

1. $A \to BC$ für $B, C \in N$,
2. $A \to x$ für $x \in T$, oder
3. $S \to \varepsilon$

und S kommt nicht auf der rechten Seite einer Produktion vor.

Insbesondere hat also eine Grammtik in Chomsky Normalform fast keine ε-Produktionen.

Lemma 2.26. *Für jede kontextfreie Sprache L existiert eine Grammatik G in Chomsky Normalform so dass* $\mathrm{L}(G) = L$.

Beweis. Nach Lemma 2.24 existiert eine gekürzte Grammatik $G_0 = (N_0, T, P_0, S)$ ohne Umbenennungen und fast ohne ε-Produktionen mit $\mathrm{L}(G_0) = L$ so dass S nicht auf der rechten Seite von P_0 vorkommt. Sei $N_1 = N_0 \cup \{A_x \mid x \in T\}$. Für $\alpha \in N_0 \cup T$ sei

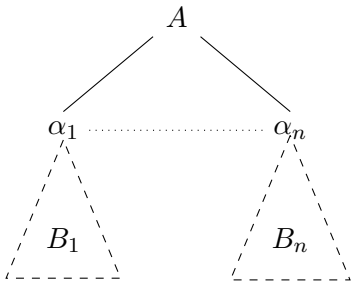
$$\hat{\alpha} = \begin{cases} A_\alpha & \text{falls } \alpha \in T \\ \alpha & \text{falls } \alpha \in N_0 \end{cases}$$

Sei weiters

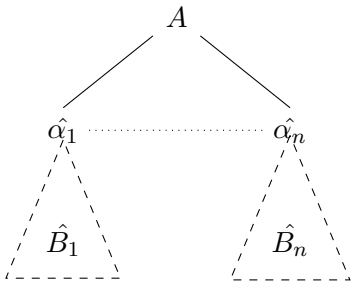
$$\begin{aligned} P_1 = {} & \{A_x \to x \mid x \in T\} \cup \\ & \{A \to x \mid A \to x \in P, x \in T\} \cup \\ & \{A \to \hat{\alpha_1} \cdots \hat{\alpha_n} \mid A \to \alpha_1 \cdots \alpha_n \in P, \alpha_1, \ldots, \alpha_n \in N_0 \cup T, n \geqslant 2\} \end{aligned}$$

[1] benannt nach Noam Chomsky

und $G_1 = (N_1, T, P_1, S)$. Dann definieren wir $f : \mathcal{B}(G_0) \to \mathcal{B}(G_1)$ durch Ersetzung von



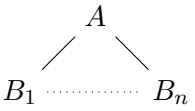
mit $n \geqslant 2$ durch



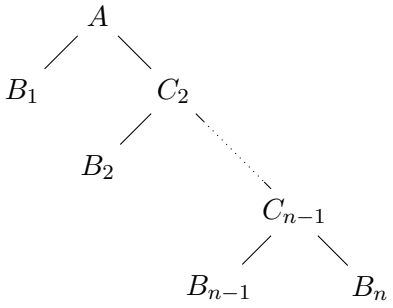
wobei $\hat{B}_i = B_i$ falls $\alpha_i \in N_0$ und $\hat{B}_i$ ist $A_{\alpha_i} \Longrightarrow \alpha_i$ falls $\alpha_i \in T$. Dann ist $f(\mathcal{B}_w^S(G_0)) = \mathcal{B}_w^S(G_1)$. In G_1 sind also alle Produktionen von der Form $A \to x$, $S \to \varepsilon$ oder $A \to B_1 \cdots B_n$. Wir erhalten G_2 aus G_1 indem wir jede Produktion der Form $A \to B_1 \cdots B_n$ ersetzen durch

$$A \to B_1 C_2, C_2 \to B_2 C_3, \ldots, C_{n-1} \to B_{n-1} B_n$$

wobei $C_2, \ldots, C_{n-1}$ neue Nichterminalsymbole sind. Dann definieren wir $g : \mathcal{B}(G_1) \to \mathcal{B}(G_2)$ durch Ersetzung von



durch



Nun ist $g(\mathcal{B}_w^S(G_1)) = \mathcal{B}_w^S(G_2)$. Damit ist $g(f(\mathcal{B}_w^S(G_0)) = \mathcal{B}_w^S(G_2)$ und Anwendung von Lemma 2.20 auf $g \circ f$ liefert das Resultat. □

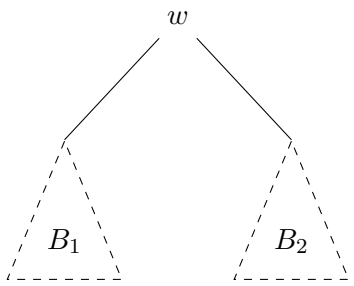
Definition 2.27. Die *Höhe* eines Baumes ist die maximale Anzahl von Kanten auf einem Pfad von der Wurzel zu einem Blatt.

Lemma 2.28. *Sei $G = (N, T, P, S)$ eine Grammatik in Chomsky Normalform, sei B ein G-Ableitungsbaum der Höhe h eines Wortes w, dann ist $|w| \leqslant 2^{h-1}$.*

Beweis. Wir gehen mit Induktion nach h vor. Falls $h = 1$ dann ist B von der Form

für ein $A \in N$ und ein $x \in T$ oder von der Form

In beiden Fällen gilt $|w| \leqslant 1$. Im Induktionsschritt ist B von der Form



wobei, für $i = 1, 2$, B_i ein Ableitungsbaum der Höhe $h-1$ für w_i ist und $w = w_1 w_2$. Dann ist $|w| = |w_1 w_2| \leqslant^{\text{IH}} 2^{h-2} + 2^{h-2} = 2^{h-1}$. □

Lemma 2.29 (Schleifensatz (pumping lemma) für kontextfreie Sprachen)**.** *Sei L eine kontextfreie Sprache. Dann gibt es ein $n \in \mathbb{N}$ so dass jedes $w \in L$ mit $|w| \geqslant n$ geschrieben werden kann als $w = v_1 v_2 v_3 v_4 v_5$ wobei*

1. *$v_2 v_4 \neq \varepsilon$,*
2. *$|v_2 v_3 v_4| \leqslant n$, und*
3. *für alle $k \geqslant 0$ ist auch $v_1 v_2^k v_3 v_4^k v_5 \in L$.*

Beweis. Sei $G = (N, T, P, S)$ eine Grammatik für L in Chomsky Normalform, sei $n = 2^{|N|}$ und sei $w \in L$ mit $|w| \geqslant n$, dann $|w| > 2^{|N|-1}$. Sei B ein G-Ableitungsbaum für w, dann hat B nach Lemma 2.28 nicht Höhe $\leqslant |N|$, also ist die Höhe von B mindestens $|N| + 1$. Seien $A_1, \ldots, A_{|N|+1}$ die $|N| + 1$ letzten Nichterminalsymbole auf einem längsten Pfad in B. Dann existieren $i, j \in \{1, \ldots, |N|+1\}$, $i < j$ mit $A_i = A_j = A$. Als Ableitung kann B dann geschrieben werden als

$$S \Longrightarrow_G^* v_1 A v_5 \Longrightarrow_G^* v_1 v_2 A v_4 v_5 \Longrightarrow_G^* v_1 v_2 v_3 v_4 v_5.$$

Damit existiert für alle $k \geqslant 0$ auch die Ableitung $S \Longrightarrow_G^* v_1 v_2^k v_3 v_4^k v_5 \in L$. Der Teilbaum von B mit Wurzel A_i ist ein Ableitungsbaum mit Höhe $\leqslant |N| + 1$ und damit $|v_2 v_3 v_4| \leqslant 2^{|N|} = n$. Weiters gibt es in G keine ε-Produktionen außer möglicherweise $S \to \varepsilon$ und in diesem Fall kommt S nicht auf der rechten Seite einer Produktion vor. Deswegen ist $v_2 v_4 \neq \varepsilon$. □

Beispiel 2.30. $L = \{a^k b^k c^k \mid k \geqslant 0\}$ ist nicht kontextfrei. Sei n für L wie im pumping Lemma für kontextfreie Sprachen und sei $w = a^n b^n c^n$. Dann kann w als $w = v_1 v_2 v_3 v_4 v_5$ geschrieben werden wobei $|v_2 v_3 v_4| \leqslant n$. Deshalb kann $v_2 v_3 v_4$ nicht sowohl a als auch c enthalten. Angenommen $v_2 v_3 v_4$ enthält a nicht, dann ist $w' = v_1 v_2^2 v_3 v_4^2 v_5 \in L$ wobei $w' = a^n b^{n'} c^{n''}$ mit $n', n'' \geqslant n$ und: $n' > n$ oder $n'' > n$. Widerspruch. Der Fall wo $v_2 v_3 v_4$ den Buchstaben c nicht enthält ist analog.

Korollar 2.31. *Die kontextfreien Sprachen sind nicht unter Durchschnitt abgeschlossen.*

Beweis. Sei $L_1 = \{a^n b^n c^k \mid k, n \geqslant 0\}$ und $L_2 = \{a^n b^k c^k \mid n, k \geqslant 0\}$. Wir haben bereits in Beispiel 2.10 gesehen, dass $L_1' = \{a^n b^n \mid n \geqslant 0\}$ kontextfrei ist. Die Sprache $L_1'' = \{c\}^*$ ist regulär und damit auch kontextfrei. Da $L_1 = L_1' \cdot L_1''$ ist auch L_1 kontextfrei. Analog kann man zeigen, dass auch L_2 kontextfrei ist. Nun gilt aber

$$L_1 \cap L_2 = \{a^k b^l c^m \mid k, l, m \geqslant 0, k = l, l = m\} = \{a^k b^k c^k \mid k \geqslant 0\}.$$

Von dieser Sprache haben wir in Beispiel 2.30 gezeigt, dass sie nicht kontextfrei ist. □

Korollar 2.32. *Die kontextfreien Sprachen sind nicht unter Komplement abgeschlossen.*

Beweis. Angenommen die kontextfreien Sprachen wären unter Komplement abgeschlossen, dann wären sie wegen $L_1 \cap L_2 = (L_1^c \cup L_2^c)^c$ auch unter Durchschnitt abgeschlossen. Widerspruch. □

2.3 Inhärent mehrdeutige Sprachen

Definition 2.33. Eine kontextfreie Sprache L heißt *inhärent mehrdeutig* falls jede kontextfreie Grammatik G mit $\mathrm{L}(G) = L$ mehrdeutig ist.

Satz 2.34. *Die kontextfreie Sprache* $L = \{a^i b^j c^k \mid i = j \text{ oder } j = k\}$ *ist inhärent mehrdeutig.*

Beispiel 2.35. Für die Sprache L aus Satz 2.34 gilt offensichtlich: $L = \{a^i b^i c^k \mid i, k \in \mathbb{N}\} \cup \{a^i b^k c^k \mid i, k \in \mathbb{N}\}$. Eine Grammatik die L erzeugt ist zum Beispiel durch die folgenden Produktionen gegeben:

$$\begin{aligned} &S \to DC \mid AE \\ &D \to aDb \mid \varepsilon \qquad\qquad A \to aA \mid \varepsilon \\ &C \to cC \mid \varepsilon \qquad\qquad E \to bEc \mid \varepsilon \end{aligned}$$

Man kann sich leicht davon überzeugen, dass Wörter der Form $a^i b^i c^i$ bezüglich dieser Grammatik zwei verschiedene Ableitungsbäume erlauben.

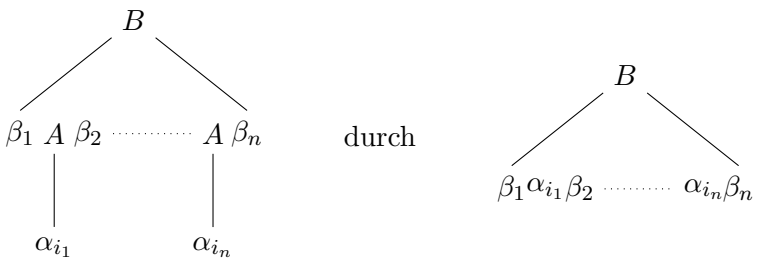
Um Satz 2.34 zu beweisen müssen wir etwas ausholen.

Lemma 2.36. *Falls* L *eine eindeutige kontextfreie Grammatik hat, dann hat* L *eine eindeutige und gekürzte kontextfreie Grammatik* $G = (N, T, P, S)$ *so dass* S *nicht auf der rechten Seite von* P *vorkommt und für alle* $A \in N \backslash \{S\}$ *Wörter* $w_1, w_2 \in T^*$ *existieren so dass* $A \Rightarrow_G^* w_1 A w_2$ *und* $w_1 w_2 \neq \varepsilon$.

Beweis. Sei $G_0 = (N_0, T, P_0, S)$ eine Grammatik mit $\mathrm{L}(G_0) = L$. Mit Lemma 2.24 können wir annehmen dass G_0 eindeutig ist, fast keine ε-Produktionen enthält, keine Umbenennungen enthält, gekürzt ist und dass S nicht auf der rechten Seite von P_0 vorkommt. Sei $A \in N_0 \backslash \{S\}$ so dass keine $w_1, w_2 \in T^*$ existieren mit $w_1 w_2 \neq \varepsilon$ und $A \Longrightarrow_{G_0}^* w_1 A w_2$. Dann gibt es auch keine $\alpha_1, \alpha_2 \in (N \cup T)^*$ mit $\alpha_1 \alpha_2 \neq \varepsilon$ und $A \Longrightarrow_{G_0}^* \alpha_1 A \alpha_2$ da G gekürzt ist und keine ε-Produktionen enthält, insbesondere gibt es keine Produktion der Form $A \to \alpha_1 A \alpha_2 \in P_0$ mit $\alpha_1 \alpha_2 \neq \varepsilon$ und, da G_0 keine Umbenennungen enthält, ist auch $A \to A \notin P_0$. Es gibt also keine Produktion die A sowohl auf der linken als auch auf der rechten Seite enthält. Sei nun $B \to \beta_1 A \beta_2 \cdots A \beta_n \in P_0$ wobei $\beta_1, \ldots, \beta_n \in (N \backslash \{A\} \cup T)^*$ und seien $A \to \alpha_1 \mid \cdots \mid \alpha_k$ alle A-Produktionen in P_1, sei

$$P_1 = P_0 \backslash \{B \to \beta_1 A \beta_2 \cdots A \beta_n\} \cup \{B \to \beta_1 \alpha_{i_1} \beta_2 \cdots \alpha_{i_{n-1}} \beta_n \mid i_1, \ldots, i_{n-1} \in \{1, \ldots, k\}\}$$

und sei $G_1 = (N_0, T, P_1, S)$. Definiere $f : \mathcal{B}(G_0) \to \mathcal{B}(G_1)$ durch Ersetzung von



dann ist $f(\mathcal{B}_w^S(G_0)) = \mathcal{B}_w^S(G_1)$ für alle $w \in T^*$. Dann ist mit Lemma 2.20 G_1 eindeutig und hat $\mathrm{L}(G_1) = L$. Weiters enthält G_1 fast keine ε-Produktionen und keine Umbenennungen. Da A in keinem der α_i vorkommt und $B \to \beta_1 A \beta_2 \cdots A \beta_n$ entfernt wird, reduziert dieser Schritt die Anzahl der Produktionen deren rechte Seite A enthält. Wiederholung dieses Schritts führt zu einer Grammatik in der A nicht mehr auf der rechten Seite vorkommt, dann kann A entfernt werden. Diese Transformation wird für alle Nichtterminale die die Bedingung des Lemmas nicht erfüllen durchgeführt. □

Weiters werden wir im Beweis von Satz 2.34 noch das folgende kombinatorische Lemma benötigen.

Lemma 2.37. *Seien $A_1, \ldots, A_n, B_1, \ldots, B_n \subseteq \mathbb{N}$, sei $S_i = A_i \times B_i$, sei $S = \bigcup_{i=1}^n S_i$. Falls $\{(a,b) \in \mathbb{N}^2 \mid a \neq b\} \subseteq S$ dann existiert ein $c \in \mathbb{N}$ so dass für alle $a \geqslant c$ gilt: $(a,a) \in S$.*

Beweis. Sei $J_0 = \{a \in \mathbb{N} \mid (a,a) \notin S\}$. Für einen Widerspruchsbeweis nehmen wir an dass J_0 unendlich ist und zeigen mit Induktion nach $k \in \{0, \ldots, n\}$ dass unendliche Mengen $J_1, \ldots, J_k$ existieren so dass $J_0 \supseteq J_1 \supseteq \cdots \supseteq J_k$ und für alle $a, b \in J_k$ gilt dass $(a,b) \notin \bigcup_{i=1}^k S_i$.

Die Induktionsbasis $k = 0$ ist trivial da $\bigcup_{i=1}^0 S_i = \varnothing$. Für $k \geqslant 1$ beobachten wir dass für alle $a \in J_{k-1}$ gilt: $a \notin A_k$ oder $a \notin B_k$, sonst wäre nämlich $(a,a) \in A_k \times B_k \subseteq S$ was mit $J_{k-1} \subseteq J_0$ auf einen Widerspruch führen würde. Folglich gibt es eine unendliche Menge $J_k \subseteq J_{k-1}$ mit $J_k \cap A_k = \varnothing$ oder $J_k \cap B_k = \varnothing$. Damit gilt für alle $a, b \in J_k$ dass $(a,b) \notin A_k \times B_k$ und, da $J_k \subseteq J_{k-1}$, auch $(a,b) \notin \bigcup_{i=1}^k S_i$.

Für $k = n$ erhalten wir also eine unendliche Menge $J_n \subseteq J_0$. Damit gibt es $a, b \in J_n$ mit $a \neq b$. Dann ist $(a,b) \notin \bigcup_{i=1}^n S_i = S$ was ein Widerspruch zur Voraussetzung ist. □

Definition 2.38. Sei $w \in T^*$, $x_1, \ldots, x_n \in T$, dann heißt $\bar{x} = x_1 \cdots x_n$ *verstreutes Teilwort von* w falls es $v_0, \ldots, v_n \in T^*$ gibt so dass $w = v_0 x_1 v_1 \cdots x_n v_n$.

Beweis von Satz 2.34. Angenommen $L = \{a^i b^j c^k \mid i = j \text{ oder } j = k\}$ hat eine eindeutige Grammatik $G = (N, \{a,b,c\}, P, S)$. Dann können wir mit Lemma 2.36 annehmen, dass G gekürzt ist und für alle $A \in N \backslash \{S\}$ Wörter $w_1, w_2 \in \{a,b,c\}^*$ existieren so dass $A \Rightarrow_G^* w_1 A w_2$ und $w_1 w_2 \neq \varepsilon$. Zunächst zeigen wir:

Behauptung 1. *Für alle $A \in N \backslash \{S\}$: Falls $A \Longrightarrow^* w_1 A w_2$ und $A \Longrightarrow^* w_3 A w_4$ dann existieren $x, y \in \{a,b,c\}$ so dass $w_1, w_3 \in x^*$ und $w_2, w_4 \in y^*$.*

Beweis. Dazu betrachten wir eine Ableitung der Form

$$\begin{aligned} S &\Longrightarrow^* v_1 A v_2 \Longrightarrow v_1 w_1 A w_2 v_2 \Longrightarrow^* v_1 w_1 w_3 A w_4 w_2 v_2 \\ &\Longrightarrow^* v_1 w_1 w_3 w_1 A w_2 w_4 w_2 v_2 \Longrightarrow^* v_1 w_1 w_3 w_1 w_3 A w_4 w_2 w_4 w_2 v_2 \Longrightarrow^* u \in L \end{aligned}$$

wobei alle v_i und w_i nur aus Terminalsymbolen bestehen. Angenommen $w_1 w_3$ enthält $x_1, x_2 \in \{a,b,c\}$ mit $x_1 \neq x_2$, dann wäre $x_1 x_2 x_1$ ein verstreutes Teilwort von $u \in L$ was ein Widerspruch ist. Ein analoges Argument zeigt dass ein $y \in \{a,b,c\}$ existiert mit $w_2 w_4 \in y^*$. □

Für $x, y \in \{a,b,c\}$ mit $x \neq y$ definieren wir

$$N_{x,y} = \{A \in N \backslash \{S\} \mid \exists\, w_1 \in x^+, w_2 \in y^+ \text{ so dass } A \Longrightarrow^* w_1 A w_2\}$$

und für $x \in \{a,b,c\}$ definieren wir

$$N_x = \{A \in N \backslash \{S\} \mid \forall w_1, w_2 \in \{a,b,c\}^* \text{ mit } A \Longrightarrow^* w_1 A w_2 \text{ gilt: } w_1, w_2 \in x^*\}.$$

Nun ist $N = \{S\} \uplus N_a \uplus N_b \uplus N_c \uplus N_{a,b} \uplus N_{a,c} \uplus N_{b,a} \uplus N_{b,c} \uplus N_{c,a} \uplus N_{c,b}$. Als nächstes zeigen wir:

Behauptung 2. $N = \{S\} \uplus N_a \uplus N_c \uplus N_{a,b} \uplus N_{b,c}$.

Beweis. Zunächst beobachten wir dass $N_{b,a} = \varnothing$. Wäre nämlich $A \in N_{b,a}$, dann gäbe es $w_1 \in b^+$ und $w_2 \in a^+$ so dass $A \Longrightarrow^* w_1 A w_2$. Damit würde die Ableitung

$$S \Longrightarrow^* v_1 A v_2 \Longrightarrow^* v_1 w_1 A w_2 v_2 \Longrightarrow^* u \in L$$

ein Wort u erzeugen das ba als verstreutes Teilwort enthält. Widerspruch. Analog zeigt man dass $N_{c,a} = \varnothing$ und $N_{c,b} = \varnothing$.

Falls eine Ableitung $S \Longrightarrow^* a^i b^j c^k \in L$ ein $A \in N_a$ enthält, dann sieht man wie beim pumping lemma dass ein $p \geqslant 1$ existiert so dass für alle $n \geqslant 0$ auch $S \Longrightarrow^* a^{i+np} b^j c^k \in L$ ist. Analoge Aussagen gelten für N_b und N_c. Falls eine Ableitung $S \Longrightarrow^* a^i b^j c^k \in L$ ein $A \in N_{a,b}$ enthält, dann sieht man wiederum wie beim pumping lemma dass es $p, q \geqslant 1$ gibt so dass für alle $n \geqslant 0$ auch $S \Longrightarrow^* a^{i+np} b^{j+nq} c^k \in L$. Analoge Aussagen gelten für $N_{a,c}$ und $N_{b,c}$.

Daraus folgt zunächst dass $N_b = \varnothing$, würde nämlich ein $A \in N_b$ in einer Ableitung $S \Longrightarrow^* a^i b^j c^k \in L$ vorkommen, dann gäbe es ein $p \geqslant 1$ so dass für alle $n \geqslant 0$ auch $a^i b^{j+np} c^k \in L$ wäre was ein Widerspruch ist. Analog folgt dass $N_{a,c} = \varnothing$: Würde nämlich $A \in N_{a,c}$ in einer Ableitung $S \Longrightarrow^* a^i b^j c^k \in L$ vorkommen, dann gäbe es $p, q \geqslant 1$ so dass für alle $n \geqslant 0$ auch $a^{i+np} b^j c^{k+nq} \in L$ was ein Widerspruch ist. □

Falls eine Ableitung $S \Longrightarrow^* a^i b^j c^k \in L$ ein $A \in N_{a,b}$ enthält, dann gibt es ja $p, q \geqslant 1$ so dass für alle $n \geqslant 0$ auch $S \Longrightarrow^* a^{i+np} b^{j+nq} c^k \in L$. Somit ist für alle bis auf ein n auch $i + np = j + nq$ und damit $i = j$ und $p = q$. Es gibt also ein $p \geqslant 1$ so dass für alle $n \geqslant 0$ auch $S \Longrightarrow^* a^{i+np} b^{j+np} c^k \in L$. Eine analoge Aussage gilt auch für $N_{b,c}$.

Weiters können gewisse Nichtterminale nicht gemeinsam vorkommen: Sei $S \Longrightarrow^* a^i b^j c^k \in L$ eine Ableitung in der sowohl ein $A \in N_a$ als auch ein $B \in N_{a,b}$ vorkommt. Dann gibt es $p, q \geqslant 1$ so dass für alle $n, m \geqslant 0$ auch $a^{i+np+mq} b^{j+mq} c^k \in L$ was ein Widerspruch ist. Sei $S \Longrightarrow^* a^i b^j c^k \in L$ eine Ableitung in der sowohl ein $A \in N_a$ als auch ein $C \in N_c$ vorkommt. Dann gibt es $p, q \geqslant 1$ so dass für alle $n, m \geqslant 0$ auch $a^{i+np} b^j c^{k+mq} \in L$ was ein Widerspruch ist. Analog zeigt man dass Nichtterminale aus N_c nicht gemeinsam mit Nichtterminalen $N_{b,c}$ in einer Ableitung auftreten können. Sei $S \Longrightarrow^* a^i b^j c^k \in L$ eine Ableitung in der sowohl ein $A \in N_{a,b}$ als auch ein $c \in N_{b,c}$ vorkommt. Dann gibt es $p, q \geqslant 1$ so dass für alle $n, m \geqslant 0$ auch $a^{i+np} b^{j+np+mq} c^{k+mq} \in L$ was ein Widerspruch ist.

Wenn $A \in N$ und $B \in N$ nicht gemeinsam in einer Ableitung auftreten können, dann existiert auch keine Produktion $A \to w \in P$ so dass B in w vorkommt. Wir definieren $P_1 = \{S \to \alpha \in P \mid \alpha \in (\{a, b, c\} \cup N_{a,b} \cup N_c)^*\}$, $P_2 = \{S \to \alpha \in P \mid \alpha \in (\{a, b, c\} \cup N_a \cup N_{b,c})^*\}$, $P_a = \{A \to w \in P \mid A \in N_a\}$ und analog für N_c, $N_{a,b}$ und $N_{b,c}$. Dann sind

$$G_1 = (N_{a,b} \cup N_c \cup \{S\}, \{a, b, c\}, P_{a,b} \cup P_c \cup P_1, S)$$
$$G_2 = (N_a \cup N_{b,c} \cup \{S\}, \{a, b, c\}, P_a \cup P_{b,c} \cup P_2, S)$$

kontextfreie Grammatiken mit $\mathrm{L}(G_1) \cup L(G_2) = L$ da S nicht auf der rechten Seite einer Produktion vorkommt. G_1 kann durch Produktionen der Form $S \to w$ für $w \in \{a, b, c\}^*$ nur endliche viele Wörter erzeugen. Jede Ableitung die mit einer Produktion der Form $S \to \alpha$ beginnt wobei α ein Nichtterminal enthält muss ein Nichtterminal aus $N_{a,b}$ oder N_c enthalten und damit ein Wort $a^i b^j c^k$ ableiten mit $i = j$. Also existiert ein $l_1 \in \mathbb{N}$ so dass

$$L(G_1) \cap \{a, b, c\}^{\geqslant l_1} \subseteq \{a^i b^j c^k \mid i = j\} \cap \{a, b, c\}^{\geqslant l_1}.$$

Mit einem analogen Argument zeigt man dass ein $l_2 \in \mathbb{N}$ existiert so dass

$$L(G_2) \cap \{a,b,c\}^{\geqslant l_2} \subseteq \{a^i b^j c^k \mid j = k\} \cap \{a,b,c\}^{\geqslant l_2}.$$

Seien nun $S \to \alpha_1, \ldots, S \to \alpha_r$ die Produktionen in P_1 mit $\alpha_l \notin \{a,b,c\}^*$. Dann betrachten wir eine Ableitung der Form $S \Longrightarrow \alpha_l \Longrightarrow_{G_1}^* a^i b^j c^k \in L$ mit $i,j,k > 0$, dann ist $i = j$ und diese Ableitung enthält ein $A \in N_{a,b}$ und ein $C \in N_c$. Folglich existieren $p, q \geqslant 1$ so dass für alle $n, m \geqslant 0$ die Ableitung $S \Longrightarrow_{G_1}^* a^{i+np} b^{i+np} c^{k+mq}$ existiert. Sei $A_l = \{i + np \mid n \in \mathbb{N}\}$, $C_l = \{k + mq \mid m \in \mathbb{N}\}$ und $T = \bigcup_{l=1}^{r} A_l \times C_l$, dann gibt es nach Lemma 2.37 ein $m_1 \in \mathbb{N}$ so dass für alle $s \geqslant m_1$ gilt: $S \Longrightarrow_{G_1}^* a^s b^s c^s$. Analog dazu zeigt man dass es ein $m_2 \in \mathbb{N}$ gibt so dass für alle $s \geqslant m_2$ gilt: $S \Longrightarrow_{G_2}^* a^s b^s c^s$. Damit erlaubt G für alle $s \geqslant \max\{m_1, m_2\}$ zwei verschiedene Ableitungsbäume für $a^s b^s c^s$. □

Für die Spezifikation formaler Sprachen, wie z.B. Programmiersprachen, durch Grammatiken ist der Aspekt der Eindeutigkeit von großer Bedeutung. Das liegt daran, dass ein Programm in einer Programmiersprache nichts anderes ist als eine Zeichenkette aus einer gewissen Menge, den wohlgeformten Programmen dieser Sprache. Seine Bedeutung enthält das Programm erst vermittels seines Ableitungsbaumes. Wenn dieser nicht eindeutig ist, dann ist das auch nicht die Bedeutung des Programms.

Kapitel 3

Berechenbarkeitstheorie

Eine zentrale Wurzel der Berechenbarkeitstheorie (die oft auch als Rekursionstheorie bezeichnet wird) ist das sogenannte[1] *Entscheidungsproblem* das von D. Hilbert 1928 gestellt wurde. Dieses Problem findet man auch in der englischsprachigen Literatur immer noch oft unter mit der ursprünglichen deutschen Bezeichnung. In aktueller Terminologie handelt es sich dabei um die folgende Frage: Gibt es einen Algorithmus, der, gegeben eine Formel φ in der Prädikatenlogik erster Stufe feststellt, ob φ eine gültige Formel ist? Ein solcher Algorithmus wäre sehr nützlich da beinahe jedes[2] mathematische Problem als eine Formel in der Prädikatenlogik erster Stufe formuliert werden kann.

Hilberts Entscheidungsproblem ist im Jahr 1936 negativ gelöst worden, und zwar unabhängig von A. Turing und A. Church: Es gibt keinen solchen Algorithmus. In einer Frage wie dieser können wir wieder eine gewisse Asymmetrie erkennen der wir in dieser Vorlesung öfters begegnen: Um zu zeigen dass ein Algorithmus mit einer gewissen Eigenschaft existiert würde es reichen, einfach einen anzugeben (und zu zeigen, dass er die geforderte Eigenschaft hat). Um allerdings zu zeigen, dass kein Algorithmus mit einer gewissen Eigenschaft existiert ist ein wesentlich komplizierteres Argument notwendig; wir benötigen dazu ein allgemeines mathematischen Modell des intuitiven Begriffs "Algorithmus" und müssen dann (in diesem Modell) zeigen, dass es keinen Algorithmus mit der geforderten Eigenschaft gibt.

In diesem Kapitel werden wir zwei unterschiedliche Modelle des intuitiven Begriffs des Algorithmus einführen und dabei die grundlegenden Begriffe, Resultate und Beweistechniken der Berechenbarkeitstheorie kennenlernen.

3.1 Partiell rekursive Funktionen

Ein Ansatz um die in einem intuitiven Sinn berechenbaren Funktionen zu definieren besteht darin, "von unten nach oben" vorzugehen, d.h. wir beginnen mit einfachen Funktionen die offensichtlich berechenbar sind und definieren dann Abschlussoperatoren die berechenbare Funktionen in berechenbare Funktionen transformieren.

Definition 3.1. Die *Basisfunktionen* sind:

1. Die konstante (nulläre) Funktion $0 \in \mathbb{N}$,

[1] Heute hat der Begriff "Entscheidungsproblem" eine allgemeinere Bedeutung.

[2] Es würde den Rahmen dieser Vorlesung sprengen, im Detail auszuführen was "beinahe jedes" hier genau bedeutet. Dazu sei an Vorlesungen über mathematische Logik verwiesen.

2. die *Nachfolgerfunktion* $\mathrm{S} : \mathbb{N} \to \mathbb{N}, x \mapsto x + 1$,

3. für alle $k \geqslant 1, 1 \leqslant i \leqslant k$, die *Projektionsfunktion* $\mathrm{P}_i^k : \mathbb{N}^k \to \mathbb{N} : (x_1, \ldots, x_k) \mapsto x_i$.

Alle Basisfunktionen sind offensichtlich berechenbar.

Definition 3.2. Sei $f : \mathbb{N}^n \to \mathbb{N}$, $g_1 : \mathbb{N}^k \to \mathbb{N}, \ldots, g_n : \mathbb{N}^k \to \mathbb{N}$. Dann ist die durch *Komposition* von f mit $g_1, \ldots, g_n$ erhaltene Funktion

$$h : \mathbb{N}^k \to \mathbb{N}, \overline{x} \mapsto f(g_1(\overline{x}), \ldots, g_n(\overline{x})).$$

Falls $f, g_1, \ldots, g_n$ berechenbar sind, dann ist auch h berechenbar: Um $h(\overline{x})$ zu berechnen, berechnen wir zunächst $y_i = g_i(\overline{x})$ für $i = 1, \ldots, n$, was nach Voraussetzung möglich ist, und danach berechnen wir $h(\overline{x}) = f(y_1, \ldots, y_n)$ was, wiederum nach Voraussetzung, möglich ist.

Definition 3.3. Sei $f : \mathbb{N}^k \to \mathbb{N}$ und $g : \mathbb{N}^{k+2} \to \mathbb{N}$. Dann erhalten wir durch *primitive Rekursion* aus f und g die Funktion $h : \mathbb{N}^{k+1} \to \mathbb{N}$, die durch

$$\begin{aligned} h(\overline{x}, 0) &= f(\overline{x}) \text{ und} \\ h(\overline{x}, y+1) &= g(\overline{x}, y, h(\overline{x}, y)) \end{aligned}$$

definiert ist.

Falls f und g berechenbar sind, dann ist das auch h. Wir gehen mit Induktion vor: Sei $\overline{x} \in \mathbb{N}^k$, $y \in \mathbb{N}$. Falls $y = 0$ ist nach Voraussetzung $f(\overline{x})$ berechenbar und damit ist das auch $h(\overline{x}, 0)$. Falls $y = y' + 1 > 0$, dann kann nach Induktionshypothese $z = h(\overline{x}, y')$ berechnet werden und damit auch $h(\overline{x}, y', z)$, da g berechenbar ist.

Definition 3.4. Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt *primitiv rekursiv* falls sie aus den Basisfunktionen mittels einer endlichen Anzahl von Kompositionen und primitiven Rekursionen aufgebaut werden kann.

Beispiel 3.5. Sei $f = \mathrm{P}_1^1 : \mathbb{N} \to \mathbb{N}$ und $g : \mathbb{N}^3 \to \mathbb{N}, (x, y, z) \mapsto z + 1$. Dann ist $g = \mathrm{S} \circ \mathrm{P}_3^3$. Mit primitiver Rekursion erhalten wir aus f und g die Funktion $h : \mathbb{N}^2 \to \mathbb{N}$ mit

$$\begin{aligned} h(x, 0) &= \mathrm{P}_1^1(x) = x, \text{ und} \\ h(x, y+1) &= g(x, y, h(x, y)) = h(x, y) + 1. \end{aligned}$$

Man sieht leicht dass h die Addition zweier natürlicher Zahlen ist, die hiermit als primitiv rekursiv nachgewiesen ist.

Viele Funktionen sind primitiv rekursiv. Wir weisen dies nun von einigen wichtigen Funktionen explizit nach.

Lemma 3.6. *Die folgenden Funktionen sind primitiv rekursiv*

1. *Addition* $(x, y) \mapsto x + y$,

2. *die konstante Funktion* $\mathrm{c}_z^k : \mathbb{N}^k \to \mathbb{N}, (x_1, \ldots, x_k) \mapsto z$,

3. *Multiplikation* $(x, y) \mapsto x \cdot y$

4. *die abgeschnittene Vorgängerfunktion* $x \mapsto \mathrm{p}(x) = \begin{cases} 0 & \text{if } x = 0 \\ x - 1 & \text{if } x > 0 \end{cases}$

5. *die abgeschnittene Subtraktion* $(x,y) \mapsto x \dot{-} y = \begin{cases} 0 & \text{if } x \leqslant y \\ x-y & \text{if } x > y \end{cases}$

6. *die charakteristische Funktion von* $\leqslant$: $(x,y) \mapsto \chi_{\leqslant}(x,y) = \begin{cases} 1 & \text{if } x \leqslant y \\ 0 & \text{if } x > y \end{cases}$

7. *die charakteristische Funktion der Gleichheit:* $(x,y) \mapsto \chi_{=}(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$

Beweis. 1. ist bereits in Beispiel 3.5 gezeigt worden. Für 2 bemerken wir zunächst dass $\mathrm{c}_z^0 = \mathrm{Cn}[\mathrm{S}, \mathrm{Cn}[\mathrm{S} \cdots \mathrm{Cn}[\mathrm{S}, 0] \cdots]]$. Für $k = 1$ verwenden wir den Pr-Operator und definieren $\mathrm{c}_z^1 = \mathrm{Pr}[\mathrm{c}_z^0, \mathrm{P}_2^2]$. Dann ist $\mathrm{c}_z^1(0) = \mathrm{c}_z^0 = z$ und $\mathrm{c}_z^1(y+1) = \mathrm{P}_2^2(y, \mathrm{c}_z^1(y)) = \mathrm{c}_z^1(y) = z$. Für $k \geqslant 2$ definieren wir einfach $\mathrm{c}_z^k = \mathrm{Cn}[\mathrm{c}_z^1, \mathrm{P}_1^k]$. Für 3 bemerken wir dass $x \cdot 0 = 0$ and $x \cdot (y+1) = x \cdot y + x$ und damit ist $\cdot = \mathrm{Pr}[f, g]$ wobei $f(x) = 0$ and $g(x,y,z) = z + x$, d.h. $f = \mathrm{c}_0^1$ und $g = \mathrm{Cn}[+, \mathrm{P}_3^3, \mathrm{P}_1^3]$. Für 4 definieren wir $\mathrm{p} = \mathrm{Pr}[0, \mathrm{P}_1^2]$. Für 5 benutzen wir eine primitiv rekursive Definition basierend auf $x \dot{-} 0 = x$ und $x \dot{-} (y+1) = \mathrm{p}(x \dot{-} y)$. Für 6 beobachten wir $\chi_{\leqslant}(x,y) = 1 \dot{-} (x \dot{-} y)$. Für 7 bemerken wir $\chi_{=}(x,y) = \chi_{\leqslant}(x,y) \cdot \chi_{\leqslant}(y,x)$. □

Die primitiv rekursiven Funktionen sind außerdem im folgenden Sinn unter Fallunterscheidung abgeschlossen.

Lemma 3.7. *Seien* $g, f_0, \ldots, f_n : \mathbb{N}^k \to \mathbb{N}$ *primitiv rekursive Funktionen. Dann ist* $h : \mathbb{N}^k \to \mathbb{N}$*, definiert durch*

$$h(\overline{x}) = \begin{cases} f_0(\overline{x}) & \text{if } g(\overline{x}) = 0 \\ f_1(\overline{x}) & \text{if } g(\overline{x}) = 1 \\ \vdots & \\ f_{n-1}(\overline{x}) & \text{if } g(\overline{x}) = n-1 \\ f_n(\overline{x}) & \text{if } g(\overline{x}) \geqslant n \end{cases}$$

ebenfalls primitiv rekursiv.

Beweis. Es gilt $h(\overline{x}) = \chi_{=}(g(\overline{x}), 0) \cdot f_0(\overline{x}) + \cdots + \chi_{=}(g(\overline{x}), n-1) \cdot f_{n-1}(\overline{x}) + \chi_{\leqslant}(n, g(\overline{x})) \cdot f_n(\overline{x})$. □

An dieser Stelle könnte man sich nun die Fragen stellen, ob wir mit dem primitiv rekursiven Funktionen alle berechenbaren Funktionen definiert haben. Die Antwort darauf ist nicht offensichtlich. Tatsächlich gibt es berechenbare Funktionen, die nicht primitiv rekursiv sind. Wir wollen nun eine derartige Funktion angeben.

Definition 3.8. Die *Ackermannfunktion*[3] ist $a : \mathbb{N}^2 \to \mathbb{N}, (p,n) \mapsto a_p(n)$ wobei

$$\begin{aligned} a_0(n) &= n+1, \\ a_{p+1}(0) &= a_p(1), \text{ und} \\ a_{p+1}(n+1) &= a_p(a_{p+1}(n)). \end{aligned}$$

Wir werden $a_p(n)$ statt $a(p,n)$ schreiben, da es oft sinnvoll ist sich die a_p als Funktionen von $\mathbb{N}$ nach $\mathbb{N}$ vorzustellen.

Lemma 3.9. *Für alle* $p, n \in \mathbb{N}$ *gilt* $a_{p+1}(n) = a_p^{n+1}(1)$.

[3] benannt nach Wilhelm Ackermann (1896-1962)

Beweis. Mit Induktion auf n: $a_{p+1}(0) = a_p(1)$, $a_{p+1}(n+1) = a_p(a_{p+1}(n)) =^{\text{IH}} a_p(a_p^{n+1}(1)) = a_p^{n+2}(1)$. □

Die Ackermannfunktion ist berechenbar. Wir gehen mit Induktion auf $p \in \mathbb{N}$ vor. Falls $p = 0$, dann ist a_p die Nachfolgerfunktion die offensichtlich berechenbar ist. Für $p+1 \in \mathbb{N}$ berechnen wir $a_{p+1}(n)$ durch $n+1$-fache Iteration von a_p auf 1.

Beispiel 3.10. a_0 ist die Nachfolgerfunktion, $a_1(n) = a_0^{n+1}(1) = n+2$, and $a_2(n) = a_1^{n+1}(1) = 1 + (n+1) \cdot 2 = 2n+3$.

Wir wollen jetzt zeigen, dass a nicht primitiv rekursiv ist. Dabei handelt es sich wiederum um eine Situation wo wir von einem konkreten Objekt (der Ackermannfunktion, einer formalen Sprache, ...) zeigen wollen, dass es keine Beschreibung einer bestimmten Art besitzt (eine Definition als primitiv rekursive Funktion, einen endlichen deterministischen Automaten,...). Anders als bei den regulären oder kontextfreien Sprachen gibt es für die primitiv rekursiven Funktionen keine einfache strukturelle Aussage wie die Schleifensätze (engl. *pumping lemmas*). Stattdessen werden wir in diesem Kontext über die Geschwindigkeit des Wachstums argumentieren. Der Schlüssel zu diesem Resultat besteht darin zu zeigen dass die Ackermannfunktion schneller wächst als jede primitiv rekursive Funktion und deshalb nicht selbst primitiv rekursiv sein kann. Wir zeigen zunächst einige grundlegende Eigenschaften der Ackermannfunktion.

Lemma 3.11. *Für alle $m, n, p, q \in \mathbb{N}$ gilt:*

1. $a_p(n) > n$,
2. $n < m$ *impliziert* $a_p(n) < a_p(m)$,
3. $p < q$ *impliziert* $a_p(n) < a_q(n)$,
4. $a_p(a_q(n)) \leqslant a_{\max\{p,q-1\}+2}(n)$.

Beweis. Wir zeigen 1. mit Induktion nach p. Der Fall $p = 0$ ist klar. Für $p+1$ gilt nach Induktionshypothese $1 < a_p(1) < a_p^2(1) < \cdots < a_p^{n+1}(1)$ und damit $a_{p+1}(n) = a_p^{n+1}(1) > n+1 > n$.

Für 2. reicht es zu zeigen dass $a_p(n) < a_p(n+1)$. Das ist klar für $p = 0$. Für $p+1$ haben wir $a_{p+1}(n+1) = a_p(a_{p+1}(n)) >^{1.} a_{p+1}(n)$.

Für 3. reicht es zu zeigen dass $a_p(n) < a_{p+1}(n)$. Wie vorhin haben wir $a_p^n(1) > n$ und damit $a_{p+1}(n) = a_p(a_p^n(1)) >^{2.} a_p(n)$.

Für 4. sei $r = \max\{p, q-1\}$. Dann ist $r \geqslant p, r+1 \geqslant q$. Wir haben

$$a_p(a_q(n)) \leqslant a_r(a_{r+1}(n)) = a_r(a_r^{n+1}(1)) = a_r^{n+2}(1),$$
$$a_{r+2}(n) = a_{r+1}^{n+1}(1) = a_{r+1}^n(a_{r+1}(1)) = a_{r+1}^n(a_r^2(1)),$$

und $a_r^{n+2}(1) \leqslant a_{r+1}^n(a_r^2(1))$. □

Wir haben $a_3(n) > 2^n$ für alle $n \in \mathbb{N}$, $a_4(n)$ ist größer als die n-fach iterierte Exponentialfunktion, usw. Wir erhalten sogar:

Lemma 3.12. *Sei $h : \mathbb{N}^m \to \mathbb{N}$ primitiv rekursiv, dann gibt es ein $p \in \mathbb{N}$ so dass für alle $\overline{x} \in \mathbb{N}^m$: $h(\overline{x}) < a_p(\max\{\overline{x}\})$*

Beweis. Da h primitiv rekursiv ist, ist es aus einer endlichen Anzahl von Kompositionen und primitiven Rekursionen zusammengesetzt. Wir zeigen das Lemma mit Induktion auf dieser Anzahl. Für die Basisfunktionen reicht es zu beobachten, dass $0 < a_0(0) = 1$, dass $s(n) = n + 1 < n + 2 = a_1(n)$ und dass $\mathrm{P}_i^m(\overline{x}) = x_i < x_i + 1 \leqslant a_0(\max\{\overline{x}\})$.

Für die Komposition sei $f : \mathbb{N}^n \to \mathbb{N}$, $g_1, \ldots, g_n : \mathbb{N}^k \to \mathbb{N}$, und $h : \mathbb{N}^k \to \mathbb{N}, \overline{x} \mapsto f(g_1(\overline{x}), \ldots, g_n(\overline{x}))$. Nach Induktionshypothese gibt es ein $p \in \mathbb{N}$ so dass $f(\overline{y}) < a_p(\max\{\overline{y}\})$ für alle $\overline{y} \in \mathbb{N}^n$ und es gibt $q_1, \ldots, q_n \in \mathbb{N}$ so dass $g_i(\overline{x}) < a_{q_i}(\max\{\overline{x}\})$ für alle $\overline{x} \in \mathbb{N}^k$ und alle $i \in \{1, \ldots, n\}$. Damit haben wir

$$h(\overline{x}) = f(g_1(\overline{x}), \ldots, g_n(\overline{x})) < a_p(\max\{g_1(\overline{x}), \ldots, g_n(\overline{x})\})$$

und, wegen der Monotonie von max und a_p,

$$< a_p\big(\max\{a_{q_1}(\max\{\overline{x}\}), \ldots, a_{q_n}(\max\{\overline{x}\})\}\big)$$

und für $q = \max\{q_1, \ldots, q_n\}$ wegen der Monotonie von a,

$$= a_p(a_q(\max\{\overline{x}\}))$$

und wegen Lemma 3.11/4.,

$$\leqslant a_{\max\{p,q-1\}+2}(\max\{\overline{x}\})$$

Für den Fall primitiver Rekursion sei $f : \mathbb{N}^k \to \mathbb{N}$, $g : \mathbb{N}^{k+2} \to \mathbb{N}$, $\overline{x} \in \mathbb{N}^k$ und $y \in \mathbb{N}$. Dann gilt $h(\overline{x}, 0) = f(\overline{x})$ und $h(\overline{x}, y + 1) = g(\overline{x}, y, h(\overline{x}, y))$. Aus der Induktionshypothese erhalten wir ein $p \in \mathbb{N}$ so dass $f(\overline{x}) < a_p(\max\{\overline{x}\})$ für alle $\overline{x} \in \mathbb{N}^k$ und ein $q \in \mathbb{N}$ so dass $g(\overline{x}, y, z) < a_q(\max\{\overline{x}, y, z\})$ für alle $\overline{x} \in \mathbb{N}^k$ und $y, z \in \mathbb{N}$. Sei $r = \max\{p, q\} + 1$. Zunächst zeigen wir dass

$$h(\overline{x}, y) < a_r(\max\{\overline{x}\} + y)$$

mit Induktion auf y. Für die Induktionsbasis haben wir $h(\overline{x}, 0) = f(\overline{x}) < a_p(\max\{\overline{x}\}) < a_r(\max\{\overline{x}\})$. Für den Induktionsschritt haben wir

$$h(\overline{x}, y + 1) = g(\overline{x}, y, h(x, y)) < a_q(\max\{\overline{x}, y, h(x, y)\}),$$

nach Induktionshypothese gilt $h(\overline{x}, y) < a_r(\max\{\overline{x}\} + y)$ und da auch $\max\{\overline{x}\}, y < a_r(\max\{\overline{x}\} + y)$

$$< a_q(a_r(\max\{\overline{x}\} + y)) \leqslant a_{r-1}(a_r(\max\{\overline{x}\} + y)) = a_r(\max\{\overline{x}\} + y + 1).$$

Um eine obere Schranke durch das Maximum statt der Summe zu erhalten, reicht es zu beobachten dass

$$h(\overline{x}, y) < a_r(\max\{\overline{x}\} + y) \leqslant a_r(2 \max\{\overline{x}, y\}) < a_r(a_2(\max\{\overline{x}, y\})) < a_{\max\{r,1\}+2}(\max\{\overline{x}, y\}).$$

□

Satz 3.13. *Die Ackermannfunktion $a : \mathbb{N}^2 \to \mathbb{N}$ ist nicht primitiv rekursiv.*

Beweis. Angenommen a wäre primitiv rekursiv. Dann gäbe es wegen Lemma 3.12 ein $p \in \mathbb{N}$ so dass für alle $x_1, x_2 \in \mathbb{N}$: $a(x_1, x_2) < a_p(\max\{x_1, x_2\})$. Aber dann wäre

$$a(p, p) < a_p(\max\{p, p\}) = a_p(p) = a(p, p),$$

Widerspruch. □

Wir sehen also, dass die primitiv rekursiven Funktionen als Modell des intuitiven Begriffs der Berechenbarkeit zu kurz greifen. Was ist uns entgangen? Eine wichtige Eigenschaft der primitiven Rekursion ist die folgende: wenn wir mit der Berechnung von $h(\overline{x}, y)$ mittels primitiver Rekursion beginnen wissen wir bereits wie oft h sich selbst aufrufen wird: y mal. In jeder Programmiersprache gibt es Konstruktionen die es erlauben, eine Rekursion oder Iteration zu beginnen *ohne* dass im Vorhinein bekannt wäre, wie oft sie wiederholt werden wird. Stattdessen wird oft eine Bedingung angegeben die festlegt wann die Rekursion bzw. Iteration beendet werden soll (wie z.B. in `while`- oder `repeat ... until`-Schleifen). Bei Verwendung einer solchen Konstruktion haben wir allerdings keine Garantie, dass die Abbruchbedingung jemals erfüllt sein wird. Es ist möglich, dass die Berechnung nicht terminiert. In diesem Fall ist der Wert der berechneten Funktion für diese Eingabedaten nicht definiert. Wir benötigen also den folgenden Begriff:

Definition 3.14. Eine *partielle Funktion* von $\mathbb{N}^n$ nach $\mathbb{N}$, geschrieben als $f : \mathbb{N}^n \hookrightarrow \mathbb{N}$, ist eine Funktion $f : D \to \mathbb{N}$ für ein $D \subseteq \mathbb{N}^n$.

Für $\overline{x} \in \mathbb{N}^n \backslash D$ sagen wir auch dass $f(\overline{x})$ nicht definiert ist. Die Operatoren der Komposition und primitiven Rekursion können auf natürliche Weise auf partielle Funktionen erweitert werden (wobei der Wert einer Funktion nur definiert ist, wenn alle Resultate die zur natürlichen Berechnung dieses Werts notwendig sind ebenfalls definiert sind).

Definition 3.15. Sei $f : \mathbb{N}^{n+1} \hookrightarrow \mathbb{N}$, dann erhalten wir durch *Minimierung* aus f die Funktion $g : \mathbb{N}^n \hookrightarrow \mathbb{N}$ wobei $g(\overline{x}) = y$ falls $f(\overline{x}, y) = 0$ und, für alle $y' < y$, $f(\overline{x}, y')$ ist definiert und $f(\overline{x}, y') \neq 0$. Fall es kein solches y gibt, dann ist $g(\overline{x})$ nicht definiert.

Falls f berechenbar ist, dann ist das auch g: wir berechnen $g(\overline{x})$ indem wir $f(\overline{x}, 0), f(\overline{x}, 1), \ldots$ berechnen bis wir ein y finden mit $f(\overline{x}, y) = 0$. Falls eine der Berechnungen von $f(\overline{x}, y')$ nicht terminiert, dann terminiert auch die Berechnung von $g(\overline{x})$ nicht und $g(\overline{x})$ ist, wie es sein sollte, nicht definiert. Falls alle Berechnungen $f(\overline{x}, y')$ terminieren aber keine davon als Ergebnis 0 liefert, dann terminiert die Berechnung von $g(\overline{x})$, wie es sein sollte, nicht.

Eine Variante des Minimierungsoperators die leicht durch diesen definiert werden kann ist der μ-Operator: Sei $R : \mathbb{N}^{k+1} \hookrightarrow \{0, 1\}$, dann schreiben wir $\mu y\, R(\overline{x}, y)$ für die partielle Funktion die ein $\overline{x} \in \mathbb{N}^k$ auf das kleinste y abbildet so dass $R(\overline{x}, y) = 1$ und $R(\overline{x}, z) = 0$ für alle $z < y$ falls ein solches y existiert und sonst undefiniert ist.

Definition 3.16. Eine partielle Funktion $f : \mathbb{N}^n \hookrightarrow \mathbb{N}$ heißt *rekursiv* falls sie aus den Basisfunktionen durch eine endliche Anzahl von Anwendungen der Operatoren Komposition, primitive Rekursion und Minimierung erhalten werden kann.

Eine partielle Funktion die rekursiv ist wird kurz auch als *partiell rekursive Funktion* bezeichnet. Die Ackermannfunktion ist partiell rekursiv. Nun finden wir uns in einer Situation wieder, die jener nach Definition der primitiv rekursiven Funktion ähnelt: Wir haben mit den partiell rekursiven Funktionen eine Klasse von Funktionen definiert, die alle im intuitiven Sinn berechenbar sind. Wie können wir sichergehen, dass wir damit alle berechenbaren Funktionen abgedeckt haben? Wir werden jetzt feststellen, dass wir von sehr unterschiedlichen Richtungen immer wieder bei der selben Klasse von Funktionen ankommen. Damit wird sich die Klasse der partiell rekursiven Funktionen, ähnlich wie im vorigen Kapitel die Klasse der regulären Sprachen, als sehr robust herausstellen.

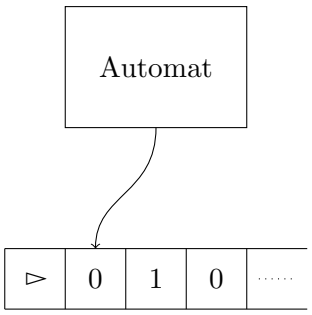


Abbildung 3.1: Struktur einer Turingmaschine

3.2 Turingmaschinen

Turingmaschinen sind ein weiteres Modell der Berechnung. Die grundlegende Idee einer Turingmaschine ist, dass ein endlicher Automat die Berechnung kontrolliert und dabei von einem unbeschränkten Speicher beliebig Gebrauch machen kann. Dieser unbeschränkte Speicher wird durch ein unendlich langes Arbeitsband modelliert, von dem jede Zelle eines von endlich vielen Zeichen enthalten kann, siehe Abbildung 3.1. Der Gebrauch ist in dem Sinn beliebig, dass jede Zelle beliebig oft geschrieben und gelesen werden kann. Um diese Schreib- und Leseoperationen durchzuführen gibt es einen Cursor, der zu jedem Zeitpunkt auf einer bestimmten Zelle des Bands positioniert ist. Die aktuelle Schreib- oder Leseoperation bezieht sich dann auf diese Zelle. Danach kann der endliche Automat den Cursor verschieben.

Wir werden hier eine Formalisierung von Turingmaschinen betrachten, wo in jeder Zelle eines der Zeichen 0, 1, ␣ oder ⊳ steht. Dabei werden wir 0 und 1 zur Binärcodierung von Daten verwenden, ␣ repräsentiert ein Leerzeichen und ⊳ markiert den Anfang des Bands. Alle Bänder die wir betrachten enthalten ␣ in allen bis auf endlichen vielen Zellen. Um die Bewegung des Cursors anzugeben schreiben wir $\leftarrow$ für die Verschiebung um eine Position nach links, $\rightarrow$ für die Verschiebung um eine Position nach rechts und $-$ für Beibehaltung der Position.

Definition 3.17. Eine Turingmaschine ist ein Tupel $M = (Q, \delta, q_0)$ wobei $q_0 \in Q$ der *Startzustand* ist und

$$\delta : \ Q \times \{0, 1, ␣, ⊳\} \longrightarrow (Q \cup \{\mathsf{ja}, \mathsf{nein}, \mathsf{fertig}\}) \times \{0, 1, ␣, ⊳\} \times \{\leftarrow, \rightarrow, -\}$$

die *Übergangsfunktion* ist und wir verlangen: falls $\delta(q, ⊳) = (q', x, d)$ für ein $q' \in Q$, dann ist $x = ⊳$ und $d \neq \leftarrow$.

Die Zustände ja, nein und fertig sind designierte Endzustände was erklärt wieso sie nicht in der Eingabe der Übergangsfunktion auftreten. Die Nebenbedingung auf δ stellt lediglich sicher, dass der Cursor nicht auf der linken Seite vom Band "herunterfallen" kann.

Definition 3.18. Sei $M = (Q, \delta, q_0)$ eine Turingmaschine. Eine *Konfiguration* von M ist ein Tupel (q, u, v) wobei $q \in Q \cup \{\mathsf{ja}, \mathsf{nein}, \mathsf{fertig}\}$, $u \in ⊳\{0, 1, ␣, ⊳\}^*$ und $v \in \{0, 1, ␣, ⊳\}^* \backslash \{0, 1, ␣, ⊳\}^* ␣$.

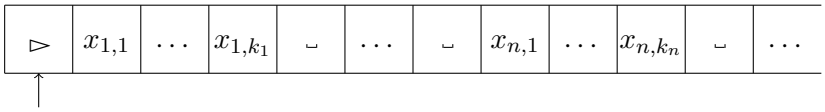
Die Konfiguration (q, u, v) wird wie folgt interpretiert: q ist der aktuelle Zustand, u ist der Inhalt des Bands auf der linken Seite inklusive Cursorposition und v ist der Inhalt des Bands auf der rechten Seite des Cursors bis die Stelle erreicht ist, ab der sich nur noch Leerzeichen auf dem Band befinden. Diese Interpretation erklärt auch die Einschränkung dass u mit einem ⊳ beginnen muss und v nicht mit einem ␣ aufhören darf.

Ein Tupel $\overline{x} = (x_1, \ldots, x_n) \in \mathbb{N}^n$ wird wie folgt auf dem Band repräsentiert: für $i = 1, \ldots, n$ sei $x_{i,1}, \ldots, x_{i,k_i} \in \{0,1\}^{k_i}$ die Binärdarstellung von x_i. Dann identifizieren wir $\overline{x}$ mit dem Wort $x_{1,1} \cdots x_{1,k_1} \sqcup \cdots \sqcup x_{n,1} \cdots x_{n,k_n}$.

Beispiel 3.19. Das Tupel $(2,3,5,7) \in \mathbb{N}^4$ wird mit dem Wort $10 \sqcup 11 \sqcup 101 \sqcup 111$ identifiziert.

Definition 3.20. Sei $\overline{x} \in \mathbb{N}^n$. Die *Initialkonfiguration für* $\overline{x}$ ist $(q_0, \rhd, \overline{x})$.

Das Band der Initialkonfiguration für $\overline{x} = (x_1, \cdots, x_n)$ kann bildlich wie folgt dargestellt werden:



Definition 3.21. Sei $M = (Q, q_0, \delta)$ eine Turingmaschine. Wir schreiben $(q,u,v) \xrightarrow{M} (q',u',v')$ falls die Konfiguration (q',u',v') in einem Schritt aus der Konfiguration (q,u,v) hervorgeht, d.h. falls für $u = u_0 x$ mit $x \in \{0,1,\sqcup,\rhd\}$ gilt: $\delta(q,x) = (q',x',r)$ und

$$(u',v') = \begin{cases} (u_0, x'v) & \text{falls } r = \leftarrow \\ (u_0x', v) & \text{falls } r = - \\ (u_0x'\sqcup, \varepsilon) & \text{falls } r = \rightarrow \text{ und } v = \varepsilon \\ (u_0x'y, v_0) & \text{falls } r = \rightarrow \text{ und } v = yv_0 \text{ mit } y \in \{0,1,\sqcup,\rhd\} \end{cases}$$

Für $k \geqslant 0$ schreiben wir $(q,u,v) \xrightarrow{M}{}^{k} (q',u',v')$ falls die Konfiguration (q',u',v') in k Schritten aus der Konfiguration (q,u,v) hervorgeht. Wir schreiben $(q,u,v) \xrightarrow{M}{}^{*} (q',u',v')$ falls es ein $k \in \mathbb{N}$ gibt so dass $(q,u,v) \xrightarrow{M}{}^{k} (q',u',v')$.

Man beachte, dass die bisher betrachteten Turingmaschinen deterministisch sind, d.h. dass für eine Turingmaschine M und eine Eingabe $\overline{x} \in \mathbb{N}^n$ genau ein $\xrightarrow{M}$-Pfad existiert, der mit der Initialkonfiguration $(q_0, \rhd, \overline{x})$ beginnt. Wir sagen, dass *die Turingmaschine M auf der Eingabe $\overline{x} \in \mathbb{N}^n$ terminiert* falls dieser Pfad endlich lang ist. Im Unterschied zu einem DFA terminiert eine Turingmaschine nicht notwendigerweise auf allen Eingaben. Eine Konfiguration der Form (ja, u, v) wird auch als ja-Konfiguration bezeichnet und eine der Form (nein, u, v) als nein-Konfiguration.

Definition 3.22. Sei $L \subseteq \{0,1\}^*$ und M eine Turingmaschine. M *entscheidet* L falls für alle $x \in \{0,1\}^*$ gilt: M terminiert auf x mit einer Endkonfiguration E_x so dass:

1. Falls $x \in L$ dann ist E_x eine ja-Konfiguration.

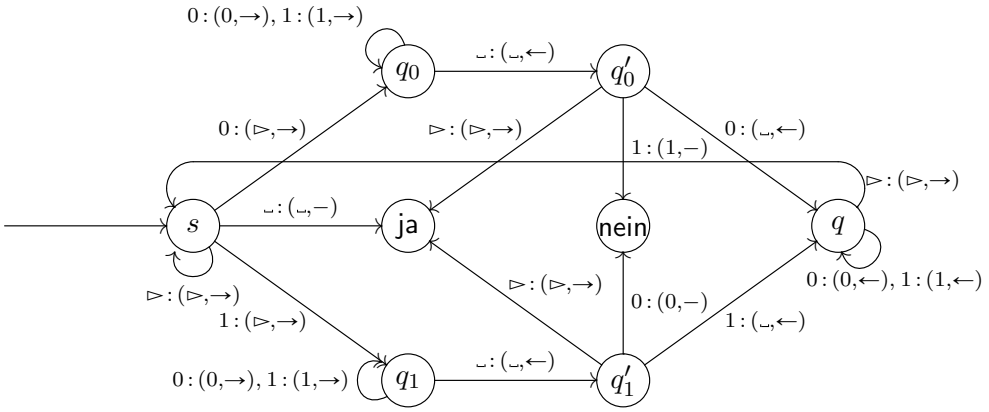
2. Falls $x \notin L$ dann ist E_x eine nein-Konfiguration.

Für Tupel $L \subseteq (\{0,1\}^*)^n$ und $\overline{x} \in \mathbb{N}^n$ wird dieser Begriff analog definiert.

Definition 3.23. Eine Sprache $L \subseteq \{0,1\}^*$ heißt *Turing-entscheidbar* falls eine Turingmaschine existiert, die L entscheidet.

Beispiel 3.24. Die folgende Turingmaschine entscheidet, ob das Eingabwort $x \in \{0,1\}^*$ ein

Palindrom ist.



Wir wollen Turingmaschinen auch dazu benutzen, Funktionen zu berechnen.

Definition 3.25. Sei M eine Turingmaschine und $k \in \mathbb{N}$. M induziert die partielle Funktion $f_{M,k} : \mathbb{N}^k \hookrightarrow \mathbb{N}$ wobei

$$f_{M,k}(\overline{x}) = \begin{cases} y & \text{falls } (q_0, \rhd, \overline{x}) \overset{M}{\to}{}^* (\mathsf{fertig}, \rhd, y) \\ \text{undef} & \text{sonst} \end{cases}$$

Man beachte dass die Funktion $f_{M,k}$ wohldefiniert ist, da die Turingmaschine M deterministisch ist.

Falls k aus dem Kontext heraus klar ist und auch sonst keine Verwechslungsgefahr vorliegt schreiben wir oft statt $f_{M,k}$ einfach nur M.

Definition 3.26. Eine partielle Funktion $f : \mathbb{N}^k \hookrightarrow \mathbb{N}$ heißt *Turing-berechenbar* falls eine Turingmaschine M existiert so dass $f = M$, d.h. für alle $\overline{x} \in \mathbb{N}^k$ gilt: $f(\overline{x})$ ist definiert genau dann wenn $M(\overline{x})$ definiert ist und in diesem Fall gilt: $f(\overline{x}) = M(\overline{x})$.

3.3 Die Church-Turing-These

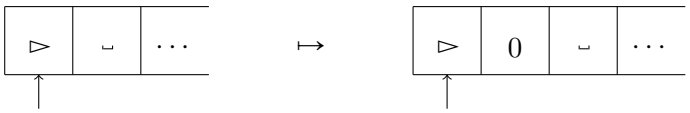
Satz 3.27. *Jede partiell rekursive Funktion ist Turing-berechenbar.*

Für den Beweis dieses Satzes werden wir mit Turingmaschinen arbeiten die k Bänder haben. Ein Turingmaschine mit $k \geqslant 1$ Bändern ist ein Tupel $M = (Q, \delta, q_0)$ wobei

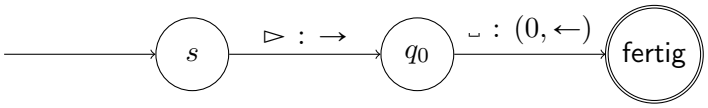
$$\delta : Q \times \{0, 1, \text{␣}, \rhd\}^k \to (Q \cup \{\mathsf{ja}, \mathsf{nein}, \mathsf{fertig}\}) \times \{0, 1, \text{␣}, \rhd\}^k \times \{\leftarrow, -, \rightarrow\}^k.$$

Ein solche Turingmaschine hat also für jedes Band einen Cursor, sie liest in jedem Schritt das aktuelle k-Tupel von Zeichen ein, schreibt mit jedem Cursor unabhängig von den anderen und bewegt jeden Cursor unabhängig von den anderen. Jede Funktion die mit einer Turingmaschine mit k Bändern berechnet werden kann kann auch mit einer Turingmaschine mit einem Band berechnet werden. Der Beweis dieser Aussage ist nicht sehr schwierig, aber etwas aufwändig weswegen wir ihn hier auslassen.

Beweis. Sei f partiell rekursiv. Dann hat f eine Operatordarstellung. Wir gehen mit Induktion auf dieser Operatordarstellung vor. Für die Induktionsbasis ist zu zeigen, dass jede der Basisfunktionen Turing-berechenbar ist. Die konstante Null entspricht der Transformation

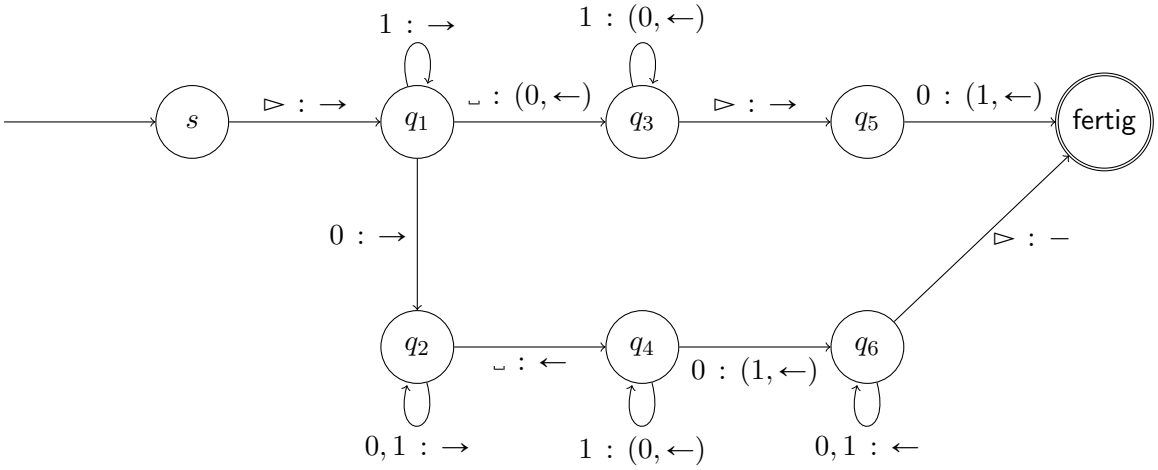


die durch die Turingmaschine[4]



berechnet wird.

Die Nachfolgerfunktion wird durch die Turingmaschine



berechnet.

Für $k \geqslant 1$ und $1 \leqslant i \leqslant k$ wird die Projektion P_i^k auf der Eingabe $(x_1, \ldots, x_k)$ durch eine Turingmaschine $M_{\mathrm{P}_i^k}$ wie folgt berechnet:

1. Überschreibe $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k$ mit ␣.
2. Bewege den Cursor zurück an den Anfang von x_i.
3. Verschiebe x_i Zeichen für Zeichen an den Anfang des Bands.
4. Bewege den Cursor auf das Startsymbol.

Eine konkrete Turingmaschine anzugeben, die diesen Algorithmus realisiert ist nicht schwierig aber technisch aufwändig.

Für den Induktionsschritt betrachten wir zunächst die Komposition. Seien $f : \mathbb{N}^n \hookrightarrow \mathbb{N}$, $g_1, \ldots, g_n : \mathbb{N}^k \hookrightarrow \mathbb{N}$. Dann berechnen wir $h = \mathrm{Cn}[f, g_1, \ldots, g_n] : \mathbb{N}^k \hookrightarrow \mathbb{N}$ durch eine Turingmaschine mit n Bändern wie folgt:

1. Kopiere die Eingabe $\overline{x}$ auf jedes der n Bänder.
2. Für $i = 1, \ldots, n$ berechne $g_i(\overline{x})$ durch die Turingmaschine M_{g_i} auf dem i-ten Band.

[4] Analog zu deterministischen endlichen Automaten vereinbaren wir bei Diagrammen zur Darstellung deterministischer Turingmaschinen dass alle nicht gezeigten Übergänge in einen Falle-Zustand führen aus dem keine Kante mehr herausführt.

3. Kopiere $g_2(\overline{x}), \ldots, g_n(\overline{x})$ auf das erste Band zu $g_1(\overline{x})$.

4. Berechne $f(g_1(\overline{x}), \ldots, g_n(\overline{x}))$ durch die Turingmaschine M_f auf dem ersten Band.

Sei $f : \mathbb{N}^k \hookrightarrow \mathbb{N}$, $g : \mathbb{N}^{k+2} \hookrightarrow \mathbb{N}$ und $h = \mathrm{Pr}[f, g] : \mathbb{N}^{k+1} \hookrightarrow \mathbb{N}$. Wir verwenden eine Turingmaschine mit 3 Bändern wie folgt: das erste Band enthält konstant die Eingabe $\overline{x}, y$ von h. Das zweite Band enthält einen Zähler z der mit 0 initialisiert wird. Das dritte Band wird unter Verwendung von M_f mit $f(\overline{x}) = h(\overline{x}, 0)$ initialisiert. In jedem Schritt wird nun unter Verwendung von M_g auf dem dritten Band aus $h(\overline{x}, z)$ der Wert $h(\overline{x}, z + 1)$ berechnet und dabei z auf dem zweiten Band entsprechend inkrementiert. Dieser Schritt wird wiederholt bis z (vom zweiten Band) gleich y (vom ersten Band) ist. Dann befindet sich die Ausgabe auf dem dritten Band.

Im Fall der Minimierung gehen wir ähnlich vor: Sei $f : \mathbb{N}^{k+1} \hookrightarrow \mathbb{N}$ und $g = \mathrm{Mn}[f] : \mathbb{N}^k \hookrightarrow \mathbb{N}$. Wir verwenden eine Turingmaschine mit 3 Bändern wie folgt: das erste Band enthält konstant die Eingabe $\overline{x}$ von g. Das zweite Band enthält einen Zähler y der mit 0 initialisiert wird. In jedem Schritt wird nun unter Verwendung von M_f auf dem dritten Band aus $\overline{x}$ (vom ersten Band) und y vom zweiten Band der Wert $f(\overline{x}, y)$ berechnet. Falls dieser Wert 0 ist befindet sich die Ausgabe auf dem zweiten Band. Falls dieser Wert nicht 0 ist wird y inkrementiert und der Schritt wird wiederholt. □

Satz 3.28. *Jede Turing-berechenbare partielle Funktion ist partiell rekursiv.*

Beweis. Um diesen Satz zu zeigen ist es notwendig Turingmaschinen, Bänder, Konfigurationen sowie Berechnungen als natürliche Zahlen zu kodieren da partiell rekursive Funktionen nur mit natürlichen Zahlen arbeiten können. Wir beginnen mit endlichen Folgen. Eine endliche Folge $x_1, \ldots, x_k$ natürlicher Zahlen kann zum Beispiel als als $\prod_{i=1}^{k} p_i^{x_i+1}$ kodiert werden wobei p_i die i-te Primzahl ist. Diese Codierung ist injektiv in dem Sinn dass jeder endlichen Folge genau ein Code zugeordnet wird. Diese Codierung ist primitiv rekursiv, in dem Sinn dass u.a. die folgenden Funktionen primitiv rekursiv sind:

1. Länge(c) gibt für Code c einer endliche Folge die Anzahl der Elemente in der Folge zurück.

2. Element(c, i) gibt für Code c einer endlichen Folge und Index $i \in \mathbb{N}$ das i-te Element der Folge zurück.

Eine Matrix wird als endliche Folge von endlichen Folgen (der selben Länge) kodiert. Eine Turingmaschine wird kodiert indem die Zustände als $q_0, \ldots, q_n$ durchnummeriert werden, die Übergangsfunktion als Matrix aufgefasst wird und ein Eintrag $\delta(q, x) = (q', x', r)$ als Folge der Länge drei kodiert wird. Ein Wort $w \in \{0, 1, \sqcup, \rhd\}^*$ wird als endliche Folge von Zeichen kodiert, jedem dieser vier Zeichen wird eine Zahl zugeordnet. Eine Konfiguration (q_i, u, v) wird kodiert als Tripel das aus i, dem Code von u und dem Code von v besteht. Eine Berechnung einer Turingmaschine M wird als Liste von Konfigurationen $K_1, \ldots, K_n$ kodiert wobei $K_i \overset{M}{\to} K_{i+1}$ für alle $i \in \{1, \ldots, n-1\}$. Diese Codierungen sind primitiv rekursiv in dem Sinn dass alle (inhaltlich vernünftigen) Operationen primitiv rekursiv sind. Insbesondere ist es durch diese Codierungen möglich zu zeigen, dass, für alle $k \in \mathbb{N}$, das folgende Prädikat $T_k \subseteq \mathbb{N}^{k+2}$ primitiv rekursiv ist:

$$
\begin{aligned}
(e, x_1, \ldots, x_k, y) \in T_k \iff\ & e \text{ ist Code einer Turingmaschine } M \text{ und} \\
& M \text{ terminiert auf Eingabe } x_1, \ldots, x_k \text{ und} \\
& y \text{ ist Code der Berechnung von } M \text{ auf } x_1, \ldots, x_k.
\end{aligned}
$$

Weiters lässt sich auf dieser Basis zeigen dass die folgende Funktion $A : \mathbb{N} \to \mathbb{N}$ primitiv rekursiv ist:

$$A(y) = z \iff y \text{ ist Code einer Berechnung deren letzte Konfiguration die Form } (\mathsf{fertig}, \rhd, z) \text{ hat}$$

Aufbauend auf diesen allgemeinen Überlegungen, sei nun M eine konkrete Turingmaschine und $k \in \mathbb{N}$. Dann berechnet M die partielle Funktion $f_{M,k} : \mathbb{N}^k \hookrightarrow \mathbb{N}$. Sei e der Code von M. Dann ist

$$f_{M,k}(x_1, \ldots, x_k) = A(\mu y\, T_k(e, x_1, \ldots, x_k, y))$$

für alle $x_1, \ldots, x_k \in \mathbb{N}$. Also ist $f_{M,k}$ partiell rekursiv. □

Ein bemerkenswerter Aspekt des obigen Beweises ist die Tatsache, dass Minimierung nur ein einziges Mal verwendet wir. Damit kann der Kleenesche Normalformensatz bewiesen werden. Wir wollen diese Richtung aber nicht weiter verfolgen. Stattdessen stellen wir fest:

Korollar 3.29. *Eine partielle Funktion ist Turing-berechenbar genau dann wenn sie partiell rekursiv ist.*

Wir haben jetzt also zwei recht unterschiedliche Formalismen gesehen, die die selbe Klasse von Funktionen beschreiben. Tatsächlich gibt es eine ganz Reihe weiterer Formalismen, von denen jeder die Klasse der partiell rekursiven Funktionen beschreibt, zum Beispiel den λ-Kalkül von A. Church dem funktionale Programmiersprachen ähneln. Ein weiterer derartiger Formalismus sind Registermaschinen, die zeitgenössischen Computern ähnlicher sind als das Turingmaschinen sind. So wie auch bei den regulären Sprachen ist eine solche Situation Indiz dafür, dass wir es mit einer wichtigen Klasse zu tun haben.

Diese Situation hat zur Church-Turing-These geführt:

Church-Turing-These

Eine partielle Funktion ist berechenbar genau dann wenn sie Turing-berechenbar ist.

Natürlich erlaubt jeder der oben betrachteten Formalismen eine äquivalente Formulierung der Church-Turing-These, z.B. als "Eine partielle Funktion ist berechenbar genau dann wenn sie partiell rekursiv ist". Wir sprechen hier von eine These und nicht von einem Satz, da ihre Aussage nicht mathematisch ist. Der Begriff "berechenbar" ist kein mathematisch definierter Begriff sondern bezieht sich auf unsere, die menschliche, Intuition für einen Algorithmus. Im Gegensatz dazu sind die betrachteten Begriffe "Turing-berechenbar", "partiell rekursiv", etc. sehr wohl mathematische Begriffe. Die, mathematisch bewiesene, Äquivalenz all dieser unterschiedlichen Begriffe rechtfertigt unser Vertrauen darin, dass wir den intuitiven Begriff der Berechenbarkeit durch diese mathematischen Begriffe korrekt beschrieben haben.

Von jetzt an werden wir nur noch über *berechenbare* (statt Turing-berechenbare) Funktionen sowie über *entscheidbare* (statt Turing-entscheidbare) Relationen sprechen.

3.4 Unentscheidbarkeit

Satz 3.30. *Es gibt unentscheidbare Sprachen.*

Beweis. Es gibt überabzählbar viele Sprachen aber nur abzählbare viele Turingmaschinen. □

Dieser Beweis ist natürlich nicht besonders befriedigend, da er kein konkretes Beispiel einer unentscheidbaren Sprache liefert. Wir werden nun ein konkretes Beispiel entwickeln. Dazu erinnern wir uns zunächst an die Codierung einer Turingmaschine als natürliche Zahl. Eine solche Codierung kann bijektiv gewählt werden indem z.B. zunächst eine beliebige Codierung gewählt wird und dann der Code e das in dieser Codierung e-te Element bezeichnet.

Definition 3.31. Sei $e \in \mathbb{N}$ dann bezeichnet TM_e die e-te Turingmaschine in der im Beweis von Satz 3.28 skizzierten Codierung.

Definition 3.32. Das *Halteproblem* ist die binäre Relation

$$H = \{(e, x) \in \mathbb{N} \times \mathbb{N} \mid \mathrm{TM}_e \text{ terminiert auf der Eingabe } x\}.$$

Satz 3.33. *Das Halteproblem ist unentscheidbar.*

Beweis. Definiere $f : \mathbb{N} \hookrightarrow \mathbb{N}$ durch

$$f(n) = \begin{cases} 0 & \text{falls } (n,n) \notin H \\ \text{undef} & \text{falls } (n,n) \in H \end{cases}$$

Angenommen es gibt eine Turingmaschine die H entscheidet, dann gibt es auch eine Turingmaschine M die f berechnet. Wir könnten M explizit konstruieren, indem wir die durch obige Definition beschriebene Berechnung formalisieren, verzichten hier aber auf diesen technischen Teil des Beweises.

Sei $M = \mathrm{TM}_e$, dann gilt:

$$\begin{aligned} (e,e) \in H &\overset{\text{Def. } f}{\iff} f(e) \text{ ist undefiniert} \\ &\overset{\text{Def. } M}{\iff} M \text{ terminiert nicht auf Eingabe } e \\ &\overset{\text{Def. } H}{\iff} (e,e) \notin H \end{aligned}$$

was ein Widerspruch ist. □

Man beachte dass es für den obigen Beweis völlig irrelevant ist, ob die Abbildung $e \mapsto \mathrm{TM}_e$ in irgendeinem Sinn konstruktiv oder berechenbar ist, es genügt dass es sich um eine Bijektion handelt.

Auf dieser Basis ließe sich nun wie folgt zeigen, dass Hilberts Entscheidungsproblem unentscheidbar ist: wir geben eine Reduktion des Halteproblems auf das Entscheidungsproblem an, d.h. eine Übersetzung des Halteproblems H in eine prädikatenlogische Formel $\varphi_H(x, y)$ so dass $(e, n) \in H$ genau dann wenn $\varphi_H(\overline{e}, \overline{n})$ gültig ist (wobei $\overline{i}$ eine Abkürzung für den Term $s^i(0)$ ist). Angenommen das Entscheidungsproblem wäre entscheidbar durch eine Turingmaschine M, dann könnte unter Zuhilfenahme von M durch diese Äquivalenz auch H entschieden werden.

Es gibt viele weitere unentscheidbare Probleme, davon auch etliche die von praktischer Relevanz in der Informatik sind. Ein bedeutendes Beispiel für ein unentscheidbares Problem aus der Mathematik ist die Lösbarkeit diophantischer Gleichungen, d.h. die folgende Frage: gegeben ein Polynom $P(x_1, \ldots, x_n)$ mit Koeffizienten aus $\mathbb{Z}$, gibt es $a_1, \ldots, a_n \in \mathbb{Z}$ so dass $P(a_1, \ldots, a_n) = 0$?

Satz 3.34. *Sei $k \in \mathbb{N}$. Dann existiert eine* universelle Turingmaschine*, d.h. eine Turingmaschine U so dass:*

$$U(e, \overline{x}) = \mathrm{TM}_e(\overline{x}) \quad \textit{für alle } e, x_1, \ldots, x_k \in \mathbb{N}$$

Beweis. Aus dem Beweis von Satz 3.28 wissen wir bereits dass die Relation

$$\begin{aligned}(c, x_1, \ldots, x_k, y) \in T_k \iff\ & c \text{ ist Code einer Turingmaschine } M \text{ und} \\ & M \text{ terminiert auf Eingabe } x_1, \ldots, x_k \text{ und} \\ & y \text{ ist Code der Berechnung von } M \text{ auf } x_1, \ldots, x_k.\end{aligned}$$

primitiv rekursiv ist. Daraus läßt sich zeigen dass auch die Relation

$$\begin{aligned}(e, x_1, \ldots, x_k, y) \in T'_k \iff\ & \mathrm{TM}_e \text{ terminiert auf Eingabe } x_1, \ldots, x_k \text{ und} \\ & y \text{ ist Code der Berechnung von } \mathrm{TM}_e \text{ auf } x_1, \ldots, x_k.\end{aligned}$$

Definiere $f : \mathbb{N}^{k+1} \hookrightarrow \mathbb{N}, (e, x_1, \ldots, x_k) \mapsto A(\mu y\, T'_k(e, x_1, \ldots, x_k, y))$. Per definitionem ist f partiell rekursiv und es gilt $f(e, \overline{x}) = \mathrm{TM}_e(\overline{x})$. Damit existiert nach Satz 3.27 eine Turingmaschine U mit $U = f$. Wir erhalten also

$$U(e, \overline{x}) = f(e, \overline{x}) = \mathrm{TM}_e(\overline{x})$$

für alle $e, x_1, \ldots, x_k \in \mathbb{N}$. □

Im obigen Beweis ist es von essentieller Bedeutung dass die Bijektion $e \mapsto \mathrm{TM}_e$ so gewählt ist, dass die Turingmaschine TM_e (in der Form $A(\mu y\, T'_k(e, \overline{x}, y))$) auf berechenbare Weise simuliert werden kann falls e als Eingabe gegeben ist. Für eine beliebige Bijektion ist das nicht der Fall.

Definition 3.35. Sei $L \subseteq \{0,1\}^*$ und $M = (Q, \delta, q_0)$ eine Turingmaschine. M *akzeptiert* L falls für alle $x \in \{0,1\}^*$ gilt:

1. Falls $x \in L$, dann terminiert M auf Eingabe x mit einer ja-Konfiguration.
2. Falls $x \notin L$, dann terminiert M auf der Eingabe x nicht.

Eine Sprache L heißt *semi-entscheidbar*[5] falls eine Turingmaschine M existiert, die L akzeptiert.

Satz 3.36. *Das Halteproblem ist semi-entscheidbar.*

Beweis. Wir ändern die universelle Turingmaschine U für $k = 1$ zu einer Turingmaschine U' so ab dass U' mit ja terminiert genau dann wenn U (in einem beliebigen Zustand und mit einem beliebigen Band) terminiert. Dann akzeptiert U' das Halteproblem. □

Jedes entscheidbare Problem ist semi-entscheidbar. Ein Problem $L \subseteq \{0,1\}^*$ ist entscheidbar genau dann wenn sowohl L als auch das Komplement von L semi-entscheidbar sind. Es gibt Probleme, die nicht semi-entscheidbar sind, z.B. das Komplement des Halteproblems.

Sei $R \subseteq \mathbb{N} \times \mathbb{N}$, dann können wir eine Menge $S \subseteq \mathbb{N}$ über existentielle Quantifikation wie folgt definieren:

$$x \in S \Leftrightarrow \exists y \text{ so dass } (x, y) \in R$$

Falls R entscheidbar ist, dann ist S semi-entscheidbar (Wir geben den Beweis dafür hier nicht an, er ist aber ähnlich zur Simulation einer Turingmaschine durch partiell rekursive Funktionen.). Außerdem gilt: für jede semi-entscheidbare Menge S gibt es eine entscheidbare Menge R so dass S aus R wie oben definiert werden kann. Auf diese Weise erlaubt existentielle Quantifikation

[5] In der Literatur üblich ist dafür vor allem die Bezeichnung "rekursiv aufzählbar" oder "berechenbar aufzählbar".

eine abstrakte Definition der semi-entscheidbaren Mengen basierend auf entscheidbaren Mengen ohne ein konkretes Maschinenmodell betrachten zu müssen.

Wir wollen zum Abschluss dieses Kapitels noch kurz auf einen Begriff der Problemreduktion eingehen.

Definition 3.37. Eine Menge $A \subseteq \mathbb{N}$ ist m-reduzierbar auf eine Menge $B \subseteq \mathbb{N}$ falls eine berechenbare Funktion $f : \mathbb{N} \to \mathbb{N}$ existiert so dass für alle $x \in \mathbb{N}$: $x \in A \Leftrightarrow f(x) \in B$. In diesem Fall schreiben wir $A \leqslant_{\mathrm{m}} B$.

Die Idee hinter dieser Definition ist dass $A \leqslant_{\mathrm{m}} B$ ausdrückt dass wir A, unter Zuhilfenahme von f lösen können wenn wir bereits wissen wie man B löst. In diesem Sinne ist B mindestens so schwer wie A. Das Subskript m kommt von der Bezeichnung "many-to-one reduction" dieser Problemreduktion. Damit ist gemeint dass mehrere $x \in A$ (durch f) auf ein $y \in B$ reduziert werden können. Die Relation $\leqslant_{\mathrm{m}}$ ist reflexiv und transitiv.

Definition 3.38. Wir definieren $K = \{x \in \mathbb{N} \mid \mathrm{TM}_x \text{ terminiert auf Eingabe } x\}$.

Bei K handelt es sich also um die Diagonale des Halteproblems. Da H semi-entscheidbar ist, ist auch k semi-entscheidbar. K ist ein schwierigstes semi-entscheidbares Problem im Sinn des folgenden Satzes.

Satz 3.39. *Sei $A \subseteq \mathbb{N}$ semi-entscheidbar. Dann ist $A \leqslant_{\mathrm{m}} K$.*

Beweis. Sei M eine Turingmaschine die A akzeptiert. Für $k \in \mathbb{N}$ definieren wir die Turingmaschine M_k wie folgt:

1. Lösche die Eingabe bis zum ersten Leerzeichen.
2. Schreibe k auf das Band.
3. Führe M aus.

'Dann gilt: $k \in A \Rightarrow M_k$ akzeptiert $\mathbb{N}$ und $k \notin A \Rightarrow M_k$ akzeptiert $\varnothing$. Sei nun $f : \mathbb{N} \to \mathbb{N}$ definiert durch $\mathrm{TM}_{f(k)} = M_k$. Dann ist f berechenbar. Wir behaupten dass $k \in A \Leftrightarrow f(k) \in K$. Falls $k \in A$ dann hält $\mathrm{TM}_{f(k)} = M_k$ auf allen $x \in \mathbb{N}$ und damit hält $\mathrm{TM}_{f(k)} = M_k$ auch auf $f(k)$, d.h. also $f(k) \in K$. Falls umgekehrt $f(k) \in K$, dann hält also $\mathrm{TM}_{f(k)} = M_k$ auf $f(k)$. Damit gibt es ein $y \in \mathbb{N}$ auf dem $\mathrm{TM}_{f(k)} = M_k$ hält. Weil aber $\mathrm{TM}_{f(k)}$ ein M_k ist, hält $\mathrm{TM}_{f(k)} = M_k$ auf allen $y \in \mathbb{N}$ und $k \in A$. □

Kapitel 4

Komplexitätstheorie

4.1 Nichtdeterministische Turingmaschinen

Definition 4.1. Eine nichtdeterministische Turingmaschine ist ein Tupel $M = (Q, \Delta, q_0)$ wobei $q_0 \in Q$ der *Startzustand* und

$$\Delta \subseteq Q \times \{0, 1, \sqcup, \rhd\} \times (Q \cup \{\mathsf{ja}, \mathsf{nein}, \mathsf{fertig}\}) \times \{0, 1, \sqcup, \rhd\} \times \{\leftarrow, \rightarrow, -\}$$

die *Übergangsrelation* ist. Wir verlangen dass für $(q, \rhd, q', s, d) \in \Delta$ gilt: $s = \rhd$ und $d \neq \leftarrow$.

Der Begriff der *Konfiguration* sowie $\xrightarrow{M}$, $\xrightarrow{M}^k$ und $\xrightarrow{M}^*$ werden wortgleich zu den deterministischen Turingmaschinen definiert. Man beachte allerdings dass nun ein gegebenes $\overline{x} \in \mathbb{N}^n$ nicht mehr einen eindeutigen $\xrightarrow{M}$-Pfad induziert.

Definition 4.2. Sei $L \subseteq \{0,1\}^*$ und $M = (Q, \Delta, q_0)$ eine nichtdeterministische Turingmaschine. Wir sagen M *entscheidet* L falls alle $\xrightarrow{M}$-Pfade endlich lang sind und für alle $x \in \{0,1\}^*$ gilt:

- Falls $x \in L$, dann gibt es eine ja-Konfiguration K so dass $(q_0, \rhd, x) \xrightarrow{M}^* K$.
- Falls $x \notin L$, dann gilt für alle Endkonfigurationen K mit $(q_0, \rhd, x) \xrightarrow{M}^* K$ dass K eine nein-Konfiguration ist.

So wie auch im Fall der NFAs ist auch hier die Existenz eines einzigen Pfades, der in einer ja-Konfiguration endet hinreichend, um die Eingabe als Element der Sprache zu erkennen. Wir werden nichtdeterministische Turingmaschinen nur für Entscheidungsprobleme benutzen, nicht um (allgemeine) Funktionen zu berechnen.

Satz 4.3. *Eine Sprache $L \subseteq \{0,1\}^*$ ist durch eine deterministische Turingmaschine entscheidbar genau dann wenn L durch eine nichtdeterministische Turingmaschine entscheidbar ist.*

Beweis. Jede deterministische Turingmaschine kann trivialerweise als nichtdeterministische Turingmaschine aufgefasst werden.

Für die andere Richtung sei $M = (Q, \Delta, q_0)$ eine nichtdeterministische Turingmaschine die L entscheidet. Für alle $(q, s) \in Q \times \{0, 1, \sqcup, \rhd\}$ sei

$$d_{q,s} = |\{(q', s', r) \in (Q \cup \{\mathsf{ja}, \mathsf{nein}, \mathsf{fertig}\}) \times \{0, 1, \sqcup, \rhd\} \times \{\leftarrow, -, \rightarrow\} \mid (q, s, q', s', r) \in \Delta\}|$$

und sei $d = \max\{d_{q,s} \mid q \in Q, s \in \{0, 1, \sqcup, \triangleright\}\}$ der Verzweigungsgrad von M.

Wir definieren eine deterministische Turingmaschine D mit 3 Bändern die M wie folgt simuliert: Auf dem ersten Band steht konstant die ursprüngliche Eingabe, auf dem zweiten Band eine Zahl in d-ärer Darstellung, das dritte Band wird als Arbeitsband für eine Variante M' von M benutzt, die ihre nichtdeterministische Wahl auf Basis der Zahl auf dem zweiten Band trifft. Falls das zweite Band nicht genug d-äre Ziffern enthält dann terminiert M' mit nein.

So wird nacheinander $n = 0, 1, 2, \ldots \in \mathbb{N}$ in d-ärer Notation auf das zweite Band geschrieben und für jedes n eine Berechnung von M' durchgeführt. Fall eine dieser Berechnungen mit ja terminiert, dann terminiert D mit ja. Falls für alle n einer festen Länge (in d-ärer Darstellung) in Endkonfigurationen führen (und D bisher nicht terminiert hat) dann terminiert D mit nein. □

Ähnlich also wie bei den endlichen Automaten ist eine Sprache durch eine deterministische Turingmaschine entscheidbar genau dann wenn sie durch ein nichtdeterministische Turingmaschine entscheidbar ist, durch den Nichtdeterminismus gewinnt man also keine Ausdrucksstärke hinzu. Wir wollen nun die Laufzeit von Turingmaschinen betrachten.

Definition 4.4. Sei M eine (deterministische oder nichtdeterministische) Turingmaschine mit Startzustand q_0 und sei $f : \mathbb{N} \to \mathbb{N}$. Wir sagen M *hat Laufzeit* f falls für alle $x \in \{0, 1\}^*$ und für alle Konfigurationen C gilt: falls $(q_0, \triangleright, x) \overset{M}{\to}{}^{k} C$, dann $k \leqslant f(|x|)$.

Definition 4.5. Eine Sprache $L \subseteq \{0, 1\}^*$ bezeichnen wir als *in deterministisch-polynomialer Zeit entscheidbar* falls eine deterministische Turingmaschine M und ein Polynom $q : \mathbb{N} \to \mathbb{N}$ existieren so dass M die Sprache L in Laufzeit q entscheidet.

L bezeichnen wir als *in nichtdeterministisch-polynomialer Zeit entscheidbar* falls es eine nichtdeterministische Turingmaschine M und ein Polynom $q : \mathbb{N} \to \mathbb{N}$ gibt so dass M die Sprache L in Laufzeit q entscheidet.

P ist die Menge der in deterministisch-polynomialer Zeit entscheidbaren Sprachen. **NP** ist die Menge der in nichtdeterministisch-polynomialer Zeit entscheidbaren Sprachen.

Da jede deterministische Turingmaschine auch eine nichtdeterministische Turingmaschine ist, ist es unmittelbar klar, dass $\mathbf{P} \subseteq \mathbf{NP}$. Der Beweis von Satz 4.3 beantwortet aber nicht die Frage nach der Inklusion in der anderen Richtung, da die Laufzeit der konstruierten deterministischen Turingmaschine exponentiell in der Laufzeit der gegebenen nichtdeterministischen Turingmaschine ist.

Die Frage ob die Inklusion in die andere Richtung, also $\mathbf{NP} \subseteq \mathbf{P}$, ebenfalls gilt ist eines der schwierigsten offenen Probleme der Mathematik. Obwohl diese Frage seit Jahrzehnten im Zentrum der Aufmerksamkeit der Forschung in der theoretischen Informatik steht, ist es bisher nicht gelungen sie zu lösen.

Man beachte, dass es sich bei nichtdeterministischer Berechnung lediglich um ein theoretisches Konzept handelt und nicht um eine realistische Form von Berechnung, in dem Sinn dass sie tatsächlich von einem Computer ausgeführt werden könnte. Das Interesse an der Klasse **NP** ist durch die Beobachtung begründet, dass eine sehr große Zahl praktisch relevanter Probleme in **NP** liegen. Wir werden bald einige Beispiele für solche Probleme sehen. Im Unterschied dazu ist man an **P** vor allem deswegen interessiert weil deterministische Berechnung in polynomialer Zeit ein recht gutes theoretisches Modell für die Klasse von Problemen die man auf einem Computer in der Praxis effizient lösen kann (zumindest für Polynome kleinen Grades und Konstanten vernünftiger Größe). Auf einer konzeptuellen Ebene ist also das Interesse an der Frage $\mathbf{P} \overset{?}{=} \mathbf{NP}$

dadurch begründet, dass man herausfinden will, ob die vielen praktisch relevanten Probleme in **NP** eine praktisch effiziente Lösung zulassen[1].

Wir werden nun ein wichtiges Beispiel nichtdeterministischer Berechnung betrachten.

Definition 4.6. Formeln in der Aussagenlogik sind induktiv wie folgt definiert. Wir beginnen mit einer abzählbar unendlichen Menge von *Atomen* $\{p_i \mid i \geqslant 1\}$ und definieren dann:

1. Jedes Atom ist eine Formel.
2. Falls φ, ψ Formeln sind, dann sind auch $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$ und $\varphi \to \psi$ Formeln.

Für eine Formel φ schreiben wir $\mathrm{L}(\varphi)$ für die Menge der Atome die in φ vorkommen. Eine Interpretation einer Formel φ ist eine Funktion $I : \mathrm{L}(\varphi) \to \{0,1\}$. Wir definieren $I(\varphi) \in \{0,1\}$ induktiv auf der Struktur der Formel φ basierend auf der üblichen Wahrheitstabelle (wobei 0 den Wahrheitswert "falsch" repräsentiert und 1 den Wahrheitswert "wahr"). Eine Formel φ heißt *erfüllbar* falls eine Interpretation I von φ existiert so dass $I(\varphi) = 1$.

Beispiel 4.7. Die Formel $\varphi = p_1 \wedge (p_1 \to \neg p_2)$ wird erfüllt durch die Interpretation $I = \{p_1 \mapsto 1, p_2 \mapsto 0\}$. Die Formel $\varphi \wedge p_2$ ist unerfüllbar.

Wir betrachten nun das folgende Entscheidungsproblem:

SAT
Eingabe: eine aussagenlogische Formel φ
Ausgabe: "ja" falls φ erfüllbar ist und "nein" sonst

Ein Entscheidungsproblem wie dieses kann durch Verwendung einer geeigneten Codierung als eine Sprache $L \subseteq \{0,1\}^*$ dargestellt werden. Wir werden das hier nicht explizit durchführen. Je nach Kontext meint man mit SAT entweder die Menge der erfüllbaren Formeln oder deren Codierung als binäre Wörter. In der Komplexitätstheorie ist die Bezeichnung "Problem" gebräuchlicher als die Bezeichnung "Sprache". Wir halten uns von nun an an diese Terminologie.

Satz 4.8. SAT $\in$ **NP**

Beweis. Um zu zeigen dass SAT $\in$ **NP** skizzieren wir wie eine nichtdeterministische Turingmaschine M die Erfüllbarkeit einer Formel in polynomialer Zeit überprüfen kann: M beginnt mit der Eingabeformel φ auf ihrem Band. Dann macht M für jedes $p_i \in \mathrm{L}(\varphi)$ eine nichtdeterministische Entscheidung p_i entweder auf 0 oder auf 1 zu setzen. Diese Ersetzung wird in der gesamten Formel durchgeführt. Auf diese Weise macht M linear viele nichtdeterministische Entscheidungen nach deren Abschluss eine Formel φ^* auf dem Eingabeband steht, die keine p_i mehr enthält. Der Wert einer solchen Formel kann in deterministisch polynomialer Zeit berechnet werden. Falls dieser Wert 1 ist, dann endet dieser Berechnungspfad von M mit ja, falls er 0 ist mit nein. □

Die Definition der semi-entscheidbaren Mengen durch existentielle Quantifikation über entscheidbare Relationen hat ein Analogon auf der Ebene der polynomialen Zeitkomplexität:

[1] Man beachte allerdings, dass es mathematisch möglich ist dass **P** = **NP** wobei allerdings der Grad und die Konstanten aller Polynome die zur deterministisch-polynomialen Simulation von **NP**-Berechnung in Frage kommen so groß sind, dass das Resultat gänzlich ohne praktische Konsequenzen bleibt.

Satz 4.9. *$S \in \mathbf{NP}$ genau dann wenn ein $R \in \mathbf{P}$ und ein Polynom $q : \mathbb{N} \to \mathbb{N}$ existieren so dass*

$$x \in S \Leftrightarrow \exists\, y \in \mathbb{N} \text{ mit } |y| < q(|x|) \text{ und } (x, y) \in R$$

wobei $|x|$ die Länge der Binärdarstellung von x ist.

Beweisskizze. Sei $R \in \mathbf{P}$ und sei S definiert durch: $x \in S$ genau dann wenn $\exists y \in \mathbb{N}$ mit $|y| < q(|x|)$ und $(x, y) \in R$. Eine **NP**-Turingmaschine M für S geht wie folgt vor: bei Eingabe x macht M zunächst $q(|x|)$ viele nichtdeterministische Entscheidungen um den Wert von y zu fixieren und berechnet danach $R(x, y)$ in deterministisch polynomialer Zeit.

In die andere Richtung, sei $S \in \mathbf{NP}$ und M eine **NP**-Turingmaschine für S. Dann ist es möglich (mit etwas technischer Arbeit) eine binäre Relation $R \in \mathbf{P}$ zu definieren, so dass y die nichtdeterministischen Entscheidungen von M repräsentiert. Nachdem die Laufzeit von M polynomial ist kann M nur polynomial viele nichtdeterministische Entscheidungen machen und damit ist also y polynomial beschränkt. □

Das obige Resultat erleichtert oft den Nachweis, dass ein gegebenes Problem in **NP** ist. Dazu reicht es nämlich, die korrekte Lösung (nichtdeterministisch) zu erraten und dann in deterministisch polynomialer Zeit zu überprüfen, ob es sich tatsächlich um eine Lösung handelt. In der Literatur bezeichnet man diese Beweisstragie auch als "guess-and-check". Wir können zum Beispiel Satz 4.8 durch einen guess-and-check Beweis wie folgt zeigen:

Beweis von Satz 4.8. SAT $\in$ **NP** da eine Formel φ erfüllbar ist genau dann wenn eine Interpretation I von φ existiert, die φ erfüllt. Die Größe von I ist linear durch die Größe von φ beschränkt und die Relation "I erfüllt φ" ist in deterministisch polynomialer Zeit berechenbar. □

4.2 NP-Vollständigkeit

Definition 4.10. Seien $L_1, L_2 \subseteq \{0,1\}^*$. Wir sagen dass L_1 *auf L_2 polynomiell reduzierbar ist*, in Symbolen $L_1 \leqslant_{\mathrm{p}} L_2$, falls es eine in deterministisch polynomialer Zeit berechenbare Funktion $f : \{0,1\}^* \to \{0,1\}^*$ gibt so dass für alle $x \in \{0,1\}^*$ gilt: $x \in L_1 \Leftrightarrow f(x) \in L_2$.

Falls $L_1 \leqslant_{\mathrm{p}} L_2$, dann ist also die Sprache L_2 (bis auf polynomielle Reduktion) mindestens so schwierig wie L_1 da eine Turingmaschine die L_2 löst verwendet werden kann um L_1 zu lösen.

Definition 4.11. Eine Sprache $L \subseteq \{0,1\}^*$ heißt **NP**-*schwer* falls für alle $L' \in \mathbf{NP}$ gilt: $L' \leqslant_{\mathrm{p}} L$. L heißt **NP**-*vollständig* falls $L \in \mathbf{NP}$ und L **NP**-schwer ist.

Wir sehen also dass die **NP**-vollständigen Probleme die, bis auf polynomielle Reduzierbarkeit, schwierigsten Probleme in **NP** sind. Damit erhält man leicht das folgende Resultat:

Lemma 4.12. *Sei L ein **NP**-vollständiges Problem. Dann ist $\mathbf{P} = \mathbf{NP}$ genau dann wenn $L \in \mathbf{P}$.*

Beweis. Falls $\mathbf{P} = \mathbf{NP}$, dann ist $L \in \mathbf{NP} = \mathbf{P}$. Für die andere Richtung sei $L \in \mathbf{P}$ und sei L auch **NP**-vollständig. Sei $L' \in \mathbf{NP}$, dann ist $L' \leqslant_{\mathrm{p}} L$ und damit gibt es eine polynomielle Reduktion f so dass $x \in L' \Leftrightarrow f(x) \in L$. Dann ist aber auch $L' \in \mathbf{P}$. Damit haben wir $\mathbf{NP} \subseteq \mathbf{P}$ erhalten. □

Wie oben erwähnt, ist das Interesse an der **P** vs. **NP** Frage vor allem dadurch motiviert, dass viele praktisch relevante Probleme in **NP** liegen. Tatsächlich sind viele von ihnen sogar **NP**-vollständig. Obiges Lemma zeigt nun dass die Frage $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ so viele äquivalente Formulierungen hat, wie es **NP**-vollständige Probleme gibt: Für jedes **NP**-vollständige Problem L ist sie äquivalent zur Frage ob es einen deterministischen polynomiellen Algorithmus für L gibt. Wir wollen über den Begriff der **NP**-Vollständigkeit hier noch eine weitere wichtige Beobachtung machen:

Lemma 4.13. *Sei L **NP**-vollständig, $L' \in \mathbf{NP}$ und $L \leqslant_{\mathrm{p}} L'$. Dann ist auch L' **NP**-vollständig.*

Beweis. Sei $L_0 \in \mathbf{NP}$, dann ist $L_0 \leqslant_{\mathrm{p}} L$. Weiters ist nach Voraussetzung $L \leqslant_{\mathrm{p}} L'$. Da $\leqslant_{\mathrm{p}}$ transitiv ist erhalten wir $L_0 \leqslant_{\mathrm{p}} L'$. Damit ist auch L' **NP**-schwer und -vollständig. □

Diese Beobachtung beschreibt die übliche Strategie um von einem neuen Problem L' zu beweisen, dass es **NP**-vollständig ist: Man zeigt einerseits dass $L' \in \mathbf{NP}$ ist und gibt andererseits eine polynomiale Reduktion von einem bereits als **NP**-vollständig bekanntem Problem L auf L' an. Damit solche Reduktionen aber eingesetzt werden können, muss zunächst einmal ein erstes Problem als **NP**-vollständig nachgewiesen werden. Das leistet der folgende Satz von Cook.

Satz 4.14 (Cook 1971)**.** SAT *ist* **NP***-vollständig.*

Beweis. In Satz 4.8 haben wir bereits gezeigt, dass SAT $\in$ **NP**. Es reicht also zu zeigen, dass SAT **NP**-schwer ist. Sei dazu $L \in \mathbf{NP}$ und $M = (Q, \Delta, q_0)$ eine nichtdeterministische Turingmaschine die L in polynomialer Zeit $P : \mathbb{N} \to \mathbb{N}$ entscheidet. Da L ein Entscheidungsproblem ist, nehmen wir o.B.d.A. an dass jeder Berechnungspfad von M entweder mit ja oder mit nein endet. Insbesondere ist fertig nicht erreichbar. Wir definieren eine Funktion φ_M die in deterministisch polynomialer Zeit berechenbar ist und die jedem $x \in \{0,1\}^*$ eine Formel $\varphi_M(x)$ zuweist die erfüllbar ist genau dann wenn M mit Eingabe x einen akzeptierenden Pfad hat, d.h. also wenn $x \in L$.

Eine für diese Konstruktion wichtige Beobachtung ist: in $P(|x|)$ Schritten kann die Turingmaschine M lediglich die ersten $P(|x|)$ Zellen ihres Bandes erreichen. Deshalb reicht es, das Verhalten von M auf diesem, endlichen, Teil des Bandes zu simulieren.

Wir definieren die folgenden aussagenlogischen Atome:

- $z_{t,q}$ für $t \in \{1, \ldots, P(|x|)\}$ und $q \in Q \cup \{\mathsf{ja}, \mathsf{nein}\}$ wobei die intendierte Interpretation von $z_{t,q}$ ist, dass sich M zum Zeitpunkt t im Zustand q befindet.
- $c_{t,i}$ für $t, i \in \{1, \ldots, P(|x|)\}$ wobei die intendierte Interpretation von $c_{t,i}$ ist, dass sich der Cursor zum Zeitpunkt t an der i-ten Zelle des Bands befindet.
- $b_{t,i,v}$ für $t, i \in \{1, \ldots, P(|x|)\}$ und $v \in \{0, 1, \rhd, \textvisiblespace\}$ wobei die intendierte Interpretation von $b_{t,i,v}$ ist, dass die i-te Zelle des Bands zum Zeitpunkt t den Wert v enthält.

Wir definieren

$$\varphi_M(x) = A_M(x) \wedge T_M(P(|x|)) \wedge U_M(P(|x|)) \wedge E_M(P(|x|))$$

wobei:

- Die Startbedingung ist

$$A_M(x) = z_{1,q_0} \wedge c_{1,1} \wedge b_{1,1,\rhd} \wedge \bigwedge_{i=1}^{|x|} b_{1,i+1,x_i} \wedge \bigwedge_{i=|x|+2}^{P(|x|)} b_{1,i,\textvisiblespace}.$$

- Die Übergangsbedingung ist $T_M(n) = T'_M(n) \wedge T''_M(n) \wedge T'''_M(n)$ wobei

$$T'_M(n) = \bigwedge_{t=1}^{n-1} \bigwedge_{i=1}^{n} \bigwedge_{q \in Q} \bigwedge_{v \in \{0,1,\rhd,\textvisiblespace\}} \left((z_{t,q} \wedge c_{t,i} \wedge b_{t,i,v}) \rightarrow \bigvee_{\substack{(q',v',d) \text{ mit} \\ (q,v,q',v',d) \in \Delta}} (z_{t+1,q'} \wedge b_{t+1,i,v'} \wedge c_{t+1,i+d}) \right)$$

wobei $i + d$, je nach Wert von d, entweder für $i + 1$, für i oder für $i - 1$ steht. Weiters definieren wir

$$T''_M(n) = \bigwedge_{t=1}^{n-1} \bigwedge_{q \in \{\mathsf{ja},\mathsf{nein}\}} \left(z_{t,q} \rightarrow \left(z_{t+1,q} \wedge \bigwedge_{i=1}^{n} (c_{t,i} \rightarrow c_{t+1,i}) \wedge \bigwedge_{i=1}^{n} \bigwedge_{v \in \{0,1,\rhd,\textvisiblespace\}} (b_{t,i,v} \rightarrow b_{t+1,i,v}) \right) \right)$$

und

$$T'''_M(n) = \bigwedge_{t=1}^{n-1} \bigwedge_{i=1}^{n} ((\neg c_{t,i} \wedge b_{t,i,v}) \rightarrow b_{t+1,i,v}).$$

- Zur Definition der Eindeutigkeitsbedingung sei

$$U(\{\varphi_1, \ldots, \varphi_k\}) = (\varphi_1 \vee \ldots \vee \varphi_k) \wedge \bigwedge_{1 \leqslant i < j \leqslant k} \neg(\varphi_i \wedge \varphi_j).$$

Dann ist $U(\{\varphi_1, \ldots, \varphi_k\})$ wahr genau dann wenn genau eines der φ_i wahr ist. Die Eindeutigkeitsbedingung ist dann definiert als

$$\begin{aligned} U_M(n) = & \bigwedge_{t=1}^{n} U(\{z_{t,q} \mid q \in Q \cup \{\mathsf{ja},\mathsf{nein}\}\}) \wedge \\ & \bigwedge_{t=1}^{n} U(\{c_{t,i} \mid 1 \leqslant i \leqslant n\}) \wedge \\ & \bigwedge_{t=1}^{n} \bigwedge_{i=1}^{n} U(\{b_{t,i,v} \mid v \in \{0,1,\rhd,\textvisiblespace\}\}) \end{aligned}$$

- Die Endbedingung ist

$$E_M(n) = z_{n,\mathsf{ja}}$$

Die Formel $\varphi_M(x)$ korrespondiert genau zur Definition der Übergangsrelation, der initialen Konfiguration, etc. Deshalb ist $\varphi_M(x)$ erfüllbar genau dann wenn M mit Eingabe x einen akzeptierenden Pfad hat.

Es ist leicht zu sehen, dass die Größe von $\varphi_M(x)$ polynomial, genauer: $O(|P(x)|^3)$ ist. Nun ist $\varphi_M(x)$ durch einen deterministischen Algorithmus berechenbar mit Zeitbedarf der linear in der Größe von $\varphi_M(x)$ ist, also kubisch und damit polynomial in der Größe von x. □

Das SAT-Problem kann auf Formeln in konjunktiver Normalform (engl. *conjunctive normal form (CNF)*) eingeschränkt werden und bleibt **NP**-vollständig. Eine aussagenlogische Formel ist in CNF falls sie von der Form $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{k_i} L_{i,j}$ ist wobei jedes $L_{i,j}$ entweder ein Atom oder ein negiertes Atom ist. In der Literatur versteht man unter dem SAT-Problem oft diese Einschränkung auf konjunktive Normalform. Das 3SAT-Problem ist SAT eingeschränkt auf konjunktive Normalformen mit $k_i \leqslant 3$ für alle $i \in \{1, \ldots, n\}$. Auch 3SAT ist bereits **NP**-vollständig.

In der Praxis spielen oft Berechnungsprobleme eine größere Rolle als Entscheidungsprobleme, so zum Beispiel

FSat
Eingabe: eine aussagenlogische Formel φ
Ausgabe: eine Interpretation I so dass $I(\varphi) = 1$ oder "unerfüllbar"

Wir betrachten aber in der Komplexitätstheorie vorzugsweise Entscheidungsprobleme. Das Verhältnis von Berechnungsproblemen zu "ihren" Entscheidungsproblemen ist oft, aber nicht immer, sehr eng. Zum Beispiel gilt für SAT und FSAT folgendes:

Definition 4.15. Eine Relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ heißt *total* falls für alle $x \in \{0,1\}^*$ ein $y \in \{0,1\}^*$ existiert so dass $(x,y) \in R$.

Wir definieren die Komplexitätsklasse

$$\mathbf{FP} = \{R \subseteq \{0,1\}^* \times \{0,1\}^* \text{ total} \mid \text{es gibt eine deterministische Turingmaschine } M \text{ mit polynomialer Laufzeit so dass } \forall x \in \{0,1\}^*\text{: } (x, M(x)) \in R\}$$

Satz 4.16. FSAT $\in$ **FP** *genau dann wenn* SAT $\in$ **P**.

Beweis. Die $\Rightarrow$-Implikation ist trivial. Für die andere Richtung nehmen wir an dass SAT $\in$ **P**. Sei eine Formel $\varphi(x_1, \ldots, x_n)$ gegeben. Wir überprüfen ob $\varphi(\overline{x})$ erfüllbar ist: Falls nicht, dann antworten wir mit "unerfüllbar", falls ja, dann ist entweder $\varphi(\top, x_2, \ldots, x_n)$ oder $\varphi(\bot, x_2, \ldots, x_n)$ erfüllbar. Wir setzen φ' auf eine erfüllbare dieser beiden Formeln, merken uns den gewählten Wert für x_1 und fangen mit φ' die nun nur noch $n-1$ Variablen enthält wieder von vorne an. Das benötigt insgesamt höchstens $2n$ SAT-Abfragen sowie polynomialen Aufwand für Formelmanipulationen. Damit ist FSAT $\in$ **FP**. □

Allerdings gibt es auch Probleme wo dieses Verhältnis weniger eng ist. So ist zum Beispiel (seit 2002) ein Algorithmus bekannt der in polynomialer Zeit feststellt, ob eine gegebene natürliche Zahl eine Primzahl ist. Allerdings ist bis heute kein Algorithmus bekannt der in polynomial Zeit eine Faktorisierung einer natürlichen Zahl berechnet und in diversen Anwendungen, vor allem in der Kryptographie, verlässt man sich auch darauf dass das in der Praxis nicht effizient möglich ist.

Wir betrachten nun ein Problem das sich strukturell stark von SAT unterscheidet, sich aber trotzdem als **NP**-vollständig herausstellen wird.

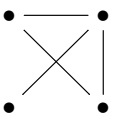
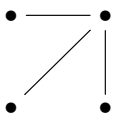
Definition 4.17. Ein *(ungerichteter) Graph* ist ein Paar $G = (V, E)$ wobei V eine Menge von *Knoten* und $E \subseteq \{\{v_1, v_2\} \mid v_1, v_2 \in V\}$ die Menge von *Kanten* ist.

Definition 4.18. Sei $G = (V, E)$ ein Graph. Ein *Pfad in* G ist eine Folge $v_1, \ldots, v_n$ von Knoten so dass $\{v_i, v_{i+1}\} \in E$ für alle $i \in \{1, \ldots, n-1\}$.

Ein *Hamiltonscher Pfad in* G ist ein Pfad der jeden Knoten $v \in V$ genau ein Mal enthält.

Hamiltonscher Pfad
Eingabe: ein endlicher Graph G
Frage: hat G einen Hamiltonschen Pfad?

Beispiel 4.19.

$G_1 =$  $G_2 =$ 

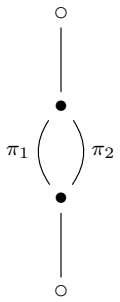
Der Graph G_1 hat einen Hamiltonschen Pfad. Der Graph G_2 hat keinen Hamiltonschen Pfad.

Satz 4.20. Hamiltonscher Pfad *ist* **NP***-vollständig.*

Beweis. Dass Hamiltonscher Pfad $\in$ **NP** ist kann durch ein guess-and-check Argument gezeigt werden: wir erraten eine Permutation der Knoten. Gegeben eine solche Permutation ist es in deterministisch polynomialer Zeit möglich festzustellen, ob es sich dabei um einen Pfad handelt.

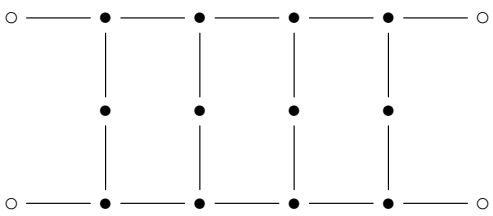
Um zu zeigen dass Hamiltonscher Pfad **NP**-schwer ist werden wir beweisen dass Sat $\leqslant_{\mathrm{p}}$ Hamiltonscher Pfad. Gegeben eine Formel $\varphi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{k_i} L_{i,j}$ in konjunktiver Normalform wollen wir in polynomialer Zeit einen Graphen G_φ berechnen so dass G_φ einen Hamiltonschen Pfad hat genau dann wenn φ erfüllbar ist. Man beachte dass eine Interpretation I die konjunktive Normalform φ erfüllt genau dann wenn für jedes $i \in \{1 \dots, n\}$ ein $j \in \{1, \dots, k_i\}$ existiert so dass $I(L_{i,j}) = 1$.

Wir werden die folgenden Bausteine benützen: den *Auswahlbaustein* für Pfade



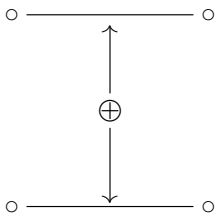
Bei Skizzen wie dieser nehmen wir an dass dieser Graph nur an den ○-Knoten mit dem Rest verbunden ist und kein Hamiltonscher Pfad existiert der an einem der •-Knoten in diesem Teil beginnt und an einem der anderen •-Knoten in diesem Teil endet. Jeder Hamiltonsche Pfad muss genau einen der beiden Pfade π_1 und π_2 enthalten. Es ist nämlich unmöglich keine der beiden auszuwählen (dann wären die beiden •-Knoten nicht im Pfad). Es ist auch unmöglich beide auszuwählen (dann wäre einer der beiden Knoten doppelt im Pfad). Wir werden diesen Baustein später dazu verwenden, eine Auswahl zwischen zwei Pfaden (und nicht zwischen zwei Kanten) zu machen.

Der *XOR-Baustein*:



Ein Hamiltonscher Pfad durchläuft diesen Graphen entweder indem er die beiden oberen ○-Knoten oder die beiden unteren ○-Knoten verbindet. Man kann leicht überprüfen dass es keine anderen Paare von ○-Knoten gibt, die eine Verbindung durch einen Hamiltonschen Pfad erlauben. Wir werden diesen Baustein verwenden, um eine XOR-Verknüpfung zwischen zwei Kanten

zu realisieren. Das wird dann geschrieben werden als:



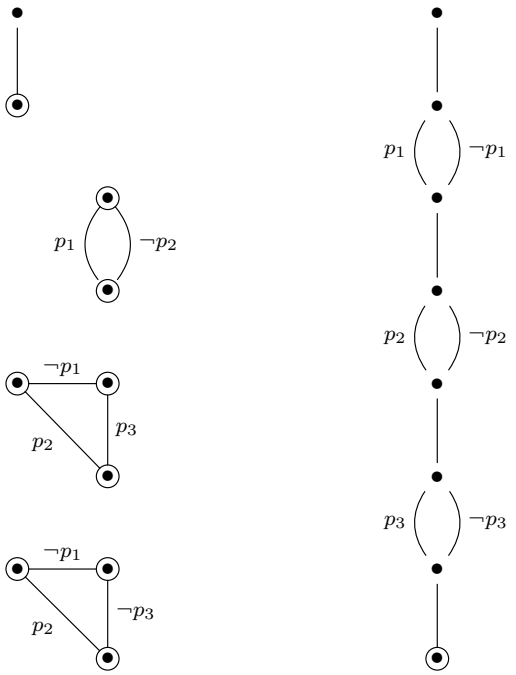
Für eine Klausel $C = L_1 \vee \ldots \vee L_n$ werden wir einen Zyklus mit n Kanten $e_1, \ldots, e_n$ in den Graphen aufnehmen wobei die Inklusion der Kante e_i im Hamiltonschen Pfad bedeuten soll dass L_i auf falsch gesetzt wird. Auf diese Weise können wir sicherstellen, dass alle Klauseln wahr sind da ein Hamiltonscher Pfad nicht alle Kanten eines Zyklus enthalten kann, da er sonst einen Knoten zwei Mal enthalten würde.

Der vierte, und letzte, Baustein ist der vollständige Graph K_n mit n Knoten. Eine für uns wichtige Eigenschaft von K_n ist: Sei $p = v_1, \ldots, v_k$ ein Pfad in K_n der keinen Knoten doppelt enthält, sei w ein Knoten der nicht in p vorkommt. Dann kann p innerhalb von K_n zu einem Hamiltonschen Pfad $v_1, \ldots, v_k, v_{k+1}, \ldots, v_n = w$ erweitert werden.

Wir beschreiben nun die Abbildung von φ auf G_φ mit einem Beispiel. Sei

$$\varphi = (p_1 \vee \neg p_2) \wedge (\neg p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee \neg p_3)$$

Dann ist der Graph G_φ



wobei, zusätzlich zu den oben eingezeichnete Kanten, noch die folgenden Kanten hinzugefügt werden:

1. Jede Kante auf der linken Seite die mit einem Literal L beschriftet ist, wird mit der eindeutigen L-Kante auf der rechten Seite mittels des XOR-Bausteins verbunden. Falls mehrere XOR-Bausteine für eine einzige Kante auf der rechten Seite verwendet werden, dann werden sie sequentiell angeordnet.

2. Alle eingeringelten Knoten werden zu einem einzigen vollständigen Graphen verbunden.

Zunächst beobachten wir dass G_φ aus φ in deterministisch polynomialer Zeit berechnet werden kann. Es bleibt zu zeigen dass G_φ einen Hamiltonschen Pfad hat genau dann wenn φ erfüllbar ist.

von links nach rechts: ein Hamiltonscher Pfad induziert eine Interpretation I durch seine Form auf der rechten Seite des Graphen. Die XOR-Bausteine erzwingen dass genau jene Kanten auf der linken Seite im Pfad liegen deren Literale durch I mit 0 interpretiert werden. Nachdem der Pfad Hamiltonsch ist kann keine der Klauselkreise alle Kanten enthalten. Somit gibt es in jeder Klausel mindestens ein auf 1 gesetztes Literal und damit ist $I(\varphi) = 1$.

von rechts nach links: Sei I eine Interpretation mit $I(\varphi) = 1$. Dann existiert der folgende Hamiltonsche Pfad: Wir starten im Knoten rechts oben und laufen, wie durch I angegeben, durch die rechte Seite bis zum eingeringelten Knoten unten rechts. Dadurch und durch die XOR-Bausteine wird die Inklusion jener Kanten auf der linken Seite erzwungen die mit Literalen beschriftet sind, die durch I auf falsch gesetzt werden. Dabei handelt es sich weiterhin um einen Hamiltonschen Pfad, da es ja wegen $I(\varphi) = 1$ keinen Klauselkreis gibt in dem alle Kanten inkludiert werden. Die verbleibenden Kanten im vollständigen Graphen werden in beliebiger Reihenfolge besucht so dass der Pfad mit dem linken oberen Teil des Graphen beendet werden kann. □

Es gibt tausende **NP**-vollständige Probleme, zum Beispiel:

Das beschränkte Grammatik-Problem

Eingabe: ein Wort $w \in A^*$ sowie ein $k \in \mathbb{N}$

Frage: existiert eine kontextfreie Grammatik G mit $\mathrm{L}(G) = \{w\}$ und $|G| \leqslant k$?

wobei die Größe einer Grammatik $G = (N, A, P, S)$ definiert ist als $|G| = \sum_{B \to \beta \in P}(|\beta| + 1)$. Weitere Beispiele für **NP**-vollständige Probleme sind:

Das Rucksack-Problem

Eingabe: eine endliche Menge U, eine Gewichtsfunktion $w : U \to \mathbb{Q}$, eine Wertfunktion $v : U \to \mathbb{Q}$ sowie $b, k \in \mathbb{N}$

Frage: existiert ein $K \subseteq U$ so dass $\sum_{u \in K} w(u) \leqslant b$ und $\sum_{u \in K} v(u) \geqslant k$?

Das Problem des Handlungsreisenden

Eingabe: Ein endlicher Graph $G = (V, E)$, eine Gewichtsfunktion $w : E \to \mathbb{N}$ und ein $k \in \mathbb{N}$

Frage: existiert ein Hamiltonscher Kreis, d.h. ein Hamiltonscher Pfad der zu seinem ersten Knoten zurückkehrt, mit Gesamtgewicht $\leqslant k$?

Das bisher Gesagte könnte den Eindruck entstehen lassen, dass sich alle praktisch relevanten Probleme relativ leicht als in **P** liegend oder als **NP**-vollständig herausstellen. Auch wenn das für viele Probleme zutrifft, gilt es bei Weitem nicht für alle: ein bekanntes Beispiel dafür ist

Graphenisomorphismus

Eingabe: Endliche Graphen G_1 und G_2

Frage: Sind G_1 und G_2 isomorph?

Man kann durch ein guess-and-check Argument leicht zeigen, dass GRAPHENISOMORPHISMUS $\in$ **NP**. Bis heute ist allerdings weder eine polynomialer Algorithmus für GRAPHENISOMORPHISMUS bekannt noch ein Beweis der **NP**-Vollständigkeit von GRAPHENISOMORPHISMUS.

4.3 Der Zeithierachiesatz

Wir wollen nun den Zeithierachiesatz zeigen. Dieser zeigt dass recht knapp aneinanderliegende Zeitkomplexitätsklassen verschieden sind. Der Beweis dieses Satzes geht im Wesentlichen so vor wir der Beweis dafür dass das Halteproblem unentscheidbar aber semi-entscheidbar ist. Allerdings wird dafür eine zeitbeschränkte Variante des Halteproblems verwendet.

Definition 4.21. Sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion. Wir definieren

$$O(f) = \{g : \mathbb{N} \to \mathbb{N} \mid \exists c > 0, n_0 \in \mathbb{N} \text{ so dass } \forall n \geqslant n_0 \colon g(n) \leqslant c \cdot f(n)\} \text{ und}$$
$$\Omega(f) = \{g : \mathbb{N} \to \mathbb{N} \mid \exists c > 0, n_0 \in \mathbb{N} \text{ so dass } \forall n \geqslant n_0 \colon c \cdot f(n) \leqslant g(n)\}.$$

Wir definieren weiters

$$o(f) = \{g : \mathbb{N} \to \mathbb{N} \mid \forall c > 0 \exists n_0 \in \mathbb{N} \forall n \geqslant n_0 \colon g(n) \leqslant c \cdot f(n)\}.$$

Statt $g \in O(f)$ schreiben wir oft $g(n) = O(f(n))$.

Definition 4.22. Sei $f : \mathbb{N} \to \mathbb{N}$, dann schreiben wir $\mathbf{ZEIT}(f)$ für die Menge der Entscheidungsprobleme $L \subseteq \{0,1\}^*$ die von einer deterministischen Turingmaschine mit Laufzeit $O(f(n))$ gelöst werden können.

Mit dieser Notation ist zum Beispiel $\mathbf{P} = \bigcup_{k \geqslant 0} \mathbf{ZEIT}(n^k)$.

Definition 4.23. Eine Funktion $f : \mathbb{N} \to \mathbb{N}$ heißt *zeitkonstruierbar* falls $f \in \Omega(n \log n)$ und es eine mehrbändige Turingmaschine gibt die, für alle $x \in \{0,1\}^*$ bei Eingabe x in genau $f(|x|)$ Schritten terminiert.

Dabei handelt es sich um eine technische Bedingung die keine starke Einschränkung darstellt, zum Beispiel sind alle (hinreichend stark wachsenden) Polynome in $\mathbb{N}[x]$ zeitkonstruierbar.

Definition 4.24. Sei $f : \mathbb{N} \to \mathbb{N}$, dann definieren wir

$H_f = \{e \llcorner x \llcorner 1^k \mid k \in \mathbb{N}, \mathrm{TM}_e \text{ hält auf Eingabe } x \text{ in höchstens } f(|e \llcorner x \llcorner 1^k|) \text{ Schritten}\}$ und
$K_f = \{e \llcorner 1^k \mid k \in \mathbb{N}, \mathrm{TM}_e \text{ hält auf Eingabe } e \text{ in höchstens } f(|e \llcorner 1^k|) \text{ Schritten mit “nein”}\}$.

Lemma 4.25. *Seien $f, h : \mathbb{N} \to \mathbb{N}$ mit $f(n) = o(h(n))$. Dann ist $K_h \notin \mathbf{ZEIT}(f(n))$.*

Beweis. Wir nehmen an, dass eine Turingmaschine M existiert, die K_h in Zeit $t(n) = O(f(n)) = o(h(n))$ entscheidet. Sei $M = \mathrm{TM}_e$ und, für $k \in \mathbb{N}$ sei $w_k = e \llcorner 1^k$. Da $t(n) = o(h(n))$ gibt es ein k so dass $t(|w_k|) < h(|w_k|)$ ist. Wir betrachten die Berechnung von M auf w_k:

1. M hält mit “nein” $\Rightarrow^{\text{Def. } K_h}$ $w_k \in K_h \Rightarrow^{\text{Def. } M}$ M hält mit “ja”
2. M hält mit “ja” $\Rightarrow^{\text{Def. } K_h}$ $w_k \notin K_h \Rightarrow^{\text{Def. } M}$ M hält mit “ja”

Beide Fälle führen also auf einen Widerspruch. □

Wir wollen nun eine obere Schranke für die Zeitkomplexität von H_f und damit auch K_f zeigen. Dazu benötigen wir noch das folgende Lemma

Lemma 4.26. *Für jede Turingmaschine M mit k Bändern und Laufzeit $f(n)$ gibt es eine Turingmaschine M' mit einem Band, $M(x) = M'(x)$ und Laufzeit $O(f(n)^2)$.*

Beweisskizze. Abschätzung des Laufzeitbedarfs einer einfachen Simulation. □

Lemma 4.27. *Sei $f : \mathbb{N} \to \mathbb{N}$ eine zeitkonstruierbare Funktion. Dann ist $H_f \in \mathbf{ZEIT}(f(n)^4)$.*

Beweis. Wir definieren eine Turingmaschine U_f mit 4 Bändern die H_f entscheidet. U_f ist eine zeitbeschränkte Variante der universellen Maschine und geht wie folgt vor:

1. Initialisierung. Auf Band 1 steht die Eingabe $e \sqcup x \sqcup 1^k$. Wir überprüfen zunächst ob e Code einer Turingmaschine ist. Ist das nicht der Fall beenden wir mit "nein". Das ist in Zeit $O(|e|^2)$ möglich[2]. Sei nun also $\mathrm{TM}_e = (Q, \delta, q_0)$. Wir schreiben in Zeit $O(|e|)$ (die Codierung von) q_0 auf Band 2. Wir kopieren x in Zeit $O(|x|)$ auf Band 3. Wir schreiben $1^{f(|x|)}$ auf Band 4. Das benötigt Zeit $O(f(|x|))$. Insgesamt benötigt die Initialisierung also Zeit $O(|e|^2 + f(|x|))$.

2. Simulation. Wir simulieren Schritt für Schritt die Berechnung von TM_e auf der Eingabe x. Dabei wird δ auf Band 1 nachgeschlagen, der aktuelle Zustand steht auf Band 2 und Band 3 ist das Arbeitsband von TM_e. Weiters wird in jedem Schritt der Cursor auf Band 4 um eine Position nach rechts verschoben. U_f antwortet mit "ja" falls die Simultation eine ja-Konfiguration erreicht. Falls die Simulation eine nein-Konfiguration erreicht oder der Zähler auf Band 4 ein Leerzeichen erreicht, dann antwortet U_f mit "nein". Die Simulation eines Schritts ist in Zeit $O(|e|)$ möglich. Insgesamt ist die Simulation von $f(|x|)$ Schritten also in Zeit $O(|e|f(|x|)$ möglich.

Sei $n = |e| + |x| + k + 2$ die Gesamtlänge der Eingabe. Dann terminiert U_f in Zeit $O(f(n)^2)$. Damit folgt das Resultat aus Lemma 4.26. □

Satz 4.28 (Zeithierachiesatz)**.** *Seien $f, g : \mathbb{N} \to \mathbb{N}$ zeitkonstruierbare Funktionen mit $f(n) = o(g(n)^{\frac{1}{4}})$. Dann ist $\mathbf{ZEIT}(f(n)) \subset \mathbf{ZEIT}(g(n))$.*

Beweis. Offensichtlich gilt $\mathbf{ZEIT}(f(n)) \subseteq \mathbf{ZEIT}(g(n))$. Für die Striktheit der Inklusion sei $h(n) = g(n)^{\frac{1}{4}}$. Dann ist $K_h \notin \mathbf{ZEIT}(f(n))$ nach Lemma 4.25. Allerdings ist $K_h \in \mathbf{ZEIT}(h(n)^4) = \mathbf{ZEIT}(g(n))$ nach Lemma 4.27.

□

Der Zeithierachiesatz ist sogar für $f(n) = o(g(n)/\log g(n))$ wahr. Allerdings muss dazu im Beweis eine aufwändigere Simulation für die universelle Turingmaschine verwendet werden, weswegen wir hier darauf verzichten.

Derartige Separationsresultate mit ähnlichen Beweisen existieren auch für nichtdeterministische Zeitklassen sowie für Platzklassen. Es ist also, ohne allzu große Schwierigkeiten, möglich die Beziehung von Komplexitätsklassen der selben Art mit unterschiedlichen Schranken zu klären. Die Schwierigkeit des **P** vs. **NP** Problems besteht darin dass es sich um zwei Komplexitätsklassen unterschiedlicher Art handelt, deterministische Zeit vs. nichtdeterministische Zeit. Über das Verhältnis verschiedenartiger Komplexitätsklassen ist nur wenig bekannt.

[2]Das im Detail nachzuweisen ist recht aufwändig da es von der Codierung von Turingmaschinen abhängt. Wir lassen diesen Beweis an dieser Stelle aus.

4.4 Orakel

Definition 4.29. Eine Orakel-Turingmaschine ist eine Turingmaschine M mit einem zusätzlichem Band, dem *Orakelband* und drei zusätzlichen Zuständen: $q_?, q_\text{j}, q_\text{n}$.

Sei $L \subseteq \{0,1\}^*$. Die Berechnung von M mit Orakel L verläuft wie gewohnt mit der folgenden zusätzlichen Vereinbarung: Falls M in den Zustand $q_?$ übergeht und $w \in \{0,1\}^*$ auf dem Orakelband steht, dann ist der nächste Zustand q_j falls $w \in L$ und q_n falls $w \notin L$.

Für $L \subseteq \{0,1\}^*$ schreiben wir dann M^L für M mit dem konkreten Orakel L. Eine Orakel-Turingmaschine kann also die Frage ob $w \in L$ ist in einem einzigen Schritt entscheiden. Das ist natürlich als Modell von Berechnung in der Realität für nicht-triviale L nicht realistisch. Für $L \in \{0,1\}^*$ bezeichnen wir mit $\mathbf{P}^L$ die Menge aller Probleme die von einer deterministischen Orakel-Turingmaschine mit Orakel L in polynomialer Zeit entschieden werden kann und analog für $\mathbf{NP}^L$ und nichtdeterministische Orakel-Turingmaschinen.

Beispiel 4.30. $\mathbf{NP} \subseteq \mathbf{P}^{\text{SAT}}$ da SAT **NP**-vollständig ist.

Wir wollen nun zeigen dass ein Orakel A existiert so dass $\mathbf{P}^A = \mathbf{NP}^A$ und ein Orakel B so dass $\mathbf{P}^B \neq \mathbf{NP}^B$. Ein solches Resultat ist aus dem folgenden Grund interessant: Viele Beweistechniken in der Komplexitätstheorie haben die Eigenschaft dass sie auch mit einem beliebigen Orakel im Hintergrund funktionieren. So kann zum Beispiel die Simulation einer Turingmaschine durch eine universelle Turingmaschine auch mit einem Orakel gemacht werden, indem die Orakelanfragen der simulierten Maschine durch die Orakelanfragen der universellen Turingmaschine simuliert werden. Mit einer solchen Beweistechnik (alleine) kann man also das **P** vs. **NP** Problem nicht lösen da es, je nach Orakel, anders ausgehen kann.

Um dieses Resultat zu zeigen brauchen wir zunächst noch den Satz von Savitch über Platzkomplexitätsklassen.

Definition 4.31. Sei $f : \mathbb{N} \to \mathbb{N}$. Dann schreiben wir $\mathbf{SPACE}(f)$ für die Menge der Entscheidungsprobleme $L \subseteq \{0,1\}^*$ die von einer deterministischen Turingmaschine mit einem Band der Länge $O(f(n))$ gelöst werden können. Wir schreiben $\mathbf{NSPACE}(f)$ für die Menge der Entscheidungsprobleme $L \subseteq \{0,1\}^*$ die von einer nichtdeterministischen Turingmaschine mit einem Band der Länge $O(f(n))$ gelöst werden können.

Wir definieren weiters $\mathbf{PSPACE} = \bigcup_{k \geqslant 0} \mathbf{SPACE}(n^k)$ und $\mathbf{NSPACE} = \bigcup_{k \geqslant 0} \mathbf{NSPACE}(n^k)$. Da eine Turingmaschine mit Laufzeit $f(n)$ nur $f(n)$ Zellen auf ihren Band benutzen kann gilt offensichtlich $\mathbf{ZEIT}(f) \subseteq \mathbf{SPACE}(f)$. Damit ist insbesondere auch $\mathbf{P} \subseteq \mathbf{PSPACE}$. Da jede deterministische Turingmaschine als nichtdeterministische Turingmaschine aufgefasst werden kann gilt auch $\mathbf{SPACE}(f) \subseteq \mathbf{NSPACE}(f)$. Allerdings gilt auch eine sehr starke Inklusion in die andere Richtung.

Satz 4.32 (Satz von Savitch)**.** *Sei $f : \mathbb{N} \to \mathbb{N}$ mit $f(n) \geqslant n$. Dann ist* $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{SPACE}(f(n)^2)$.

Beweisskizze. Im Wesentlichen durch eine Simulation einer nichtdeterministischen Turingmaschine M die $f(n)$ Platz benötigt durch eine deterministische Turingmaschine die mit Platz $f(n)^2$ auskommt. □

Insbesondere folgt also daraus $\mathbf{NPSPACE} = \mathbf{PSPACE}$. Die Klasse **PSPACE** hat in der Komplexitätstheorie auch eine wichtige Rolle. Insbesondere gibt es auch etliche **PSPACE**-vollständige Probleme. Damit können wir nun zeigen:

Satz 4.33. *Es gibt ein Orakel A so dass* $\mathbf{P}^A = \mathbf{NP}^A$.

Beweis. $\mathbf{P}^A \subseteq \mathbf{NP}^A$ gilt für alle $A \subseteq \{0,1\}^*$. Für die andere Richtung sei A ein **PSPACE**-vollständiges Problem. Dann ist $\mathbf{NP}^A \subseteq \mathbf{NPSPACE}$ da ja $A \in \mathbf{PSPACE}$ und wir damit sowohl die Orakel-Anfragen als auch die eigentliche Berechnung einer $\mathbf{NP}^A$-Maschine in **NPSPACE** durchführen können. Weiters ist $\mathbf{NPSPACE} \subseteq \mathbf{PSPACE}$ wegen Satz 4.32. Schließlich ist $\mathbf{PSPACE} \subseteq \mathbf{P}^A$ da A **PSPACE**-schwer ist. Insgesamt ist damit $\mathbf{NP}^A \subseteq \mathbf{P}^A$ gezeigt. □

Satz 4.34. *Es gibt ein Orakel B so dass* $\mathbf{P}^B \neq \mathbf{NP}^B$.

Beweis. Für $X \subseteq \{0,1\}^*$ sei $L_X = \{1^n \mid n \in \mathbb{N}, \text{ es gibt ein } v \in X \text{ mit } |v| = n\}$. Dann ist $L_X \in \mathbf{NP}^X$, da bei Eingabe 1^n ein Wort v mit $|v| = n$ nichtdeterministisch erzeugt werden kann und dann mit einem Aufruf des Orakels für X geklärt werden kann ob $v \in X$ ist. Bei jeder anderen Eingabe wird sofort mit "nein" geantwortet.

Wir wollen nun ein $B \subseteq \{0,1\}^*$ definieren so dass $L_B \notin \mathbf{P}^B$. Seien dazu $M_1, M_2, \ldots$ alle deterministischen Orakel-Turingmaschinen die polynomiale Laufzeit haben. O.B.d.A. nehmen wir an dass M_i Laufzeit n^i hat. Seien weiters $n_0 = 0 < n_1 < n_2 < \cdots$ so dass $2^{n_i} > n_i^i$. Wir gehen in Phasen $i = 1, 2, 3, \ldots$ vor und werden in Phase i erzwingen dass M_i^B nicht L_B entscheidet. Dazu definieren wir in Phase i eine Menge $B_i \subseteq \{w \in \{0,1\}^* \mid n_{i-1} < |w| \leqslant n_i\}$ und $B = \bigcup_{i \geqslant 1} B_i$.

Der entscheidenen Punkt dabei ist: um mit Sicherheit festzustellen ob $1^n \in L_B$ ist müsste M_i für alle $y \in \{0,1\}^{n_i}$ die Orakelanfrage "$y \in B$?" stellen. Da es aber 2^{n_i} viele $y \in \{0,1\}^{n_i}$ gibt, ist das in der Laufzeit von M_i nicht möglich. M_i kann also, für die "große" Eingabe 1^{n_i}, gar keine Antwort geben die mit Sicherheit korrekt ist. Wir wollen sicherstellen dass sie falsch ist.

In Phase i gehen wir wie folgt vor: Wir wenden M_i auf 1^{n_i} an. Falls M_i die Orakelanfrage "$y \in B$?" stellt antworten wir mit:

1. "ja" falls $|y| \leqslant n_{i-1}$ und $y \in \bigcup_{j=1}^{i-1} B_j$
2. "nein" falls $|y| \leqslant n_{i-1}$ und $y \notin \bigcup_{j=1}^{i-1} B_j$
3. "nein" falls $|y| > n_{i-1}$

Falls M_i mit "ja" antwortet dann setzen wir $B_i = \varnothing$ und damit ist $1^{n_i} \notin L_B$, also antwortet M_i^B auf Eingabe 1^{n_i} falsch. Falls M_i mit "nein" antwortet, dann finden wir ein $w \in \{0,1\}^{n_i}$ für das wir noch nicht mit "nein" geantwortet haben[3] und setzen $B_i = \{w\}$. Dann ist $1^{n_i} \in L_B$ und damit antwortet M_i^B auf Eingabe 1^{n_i} falsch. □

[3] Dass ein solches w existiert kann mit einer Abschätzung der Anzahl der von M_j mit $j < i$ in 3. erzwungenen "nein"-Antworten gezeigt werden.

Kapitel 5

Abschließende Bemerkungen

Zum Abschluss wollen wir noch einige Aspekte beleuchten, die in allen Themen dieser Lehrveranstaltung eine Rolle gespielt haben und ganz gut charakterisieren worum es in der theoretischen Informatik geht.

Ein zentraler Begriff ist der des **Beschreibungsformalismus**. Wir haben es oft mit einer Menge von Objekten sowie einer Menge von Beschreibungen dieser Objekte zu tun, seien es nun DFAs oder kontextfreie Grammatiken zur Beschreibung von Sprachen oder Operatordarstellungen zur Beschreibung partieller Funktionen von $\mathbb{N}$ nach $\mathbb{N}$. Manche der Objekte (Sprachen, partielle Funktionen) können dabei durch den betrachteten Formalismus beschrieben werden, andere nicht.

Oft stellt sich heraus dass eine Klasse von Objekten durch unterschiedliche Formalismen beschrieben werden kann. So können zum Beispiel die berechenbaren Funktionen durch Turingmaschinen oder durch Operatorterme, oder durch eine Vielzahl anderer Formalismen beschrieben werden. Auf ganz ähliche Weise können die regulären Sprachen durch endliche Automaten, durch reguläre Ausdrücke, oder durch rechtslineare Grammatiken beschrieben werden. Wir sehen also dass viele Klassen beschreibbarer Objekte eine beträchtliche **Robustheit** an den Tag legen.

Ein roter Faden der sich durch alle Aussagen über die Zugehörigkeit eines Objekts zu einer beschreibbaren Klasse zieht ist die **Assymetrie**. Um zu zeigen dass sich das Objekt in der Klasse befindet gibt man einfach eine Beschreibung an. So kann man etwa zeigen dass eine Sprache L regulär ist, indem man einen DFA angibt der L akzeptiert. Oder man zeigt dass eine Funktion primitiv rekursiv ist, indem man eine Operatordarstellung dieser Funktion angibt die ohne Minimierung auskommt. Umgekehrt ist es meist viel schwieriger zu zeigen dass ein bestimmtes Objekt nicht durch einen bestimmten Formalismus beschrieben werden kann. Um solche Aussagen zu beweisen haben wir verschiedene Techniken gesehen, zum Beispiel Schleifensätze (pumping lemmas) für reguläre und kontextfreie Sprachen, Abschätzung des Wachstums totaler Funktionen um zu zeigen dass die Ackermannfunktion nicht primitiv rekursiv ist, und Diagonalisierung an verschiedenen Stellen in der Berechenbarkeitstheorie und Komplexitätstheorie.

Dabei fällt auf dass diese verschiedenen Klassen eine im Wesentlichen **lineare Struktur** haben: Jede reguläre Sprache ist kontextfrei, jede kontextfreie Sprache ist in $\mathbf{P}$, jede Sprache $L \in \mathbf{P}$ ist primitiv rekusiv, und so weiter. Zwar gibt es immer wieder lokal gewisse Ausnahmen, d.h. im Sinne der Teilmengeninklusion unvergleichbare Klassen, aber im Großen und Ganzen liegt eine lineare Struktur vor. Das rechtfertigt es von der "Stärke" eine Formalismus zu sprechen, wie dies auch im Rahmen dieser Lehrveranstaltung gelegentlich der Fall war.

Alle Aspekte die bisher in diesem Abschlusskapitel genannt wurden findet man ebenso in

stärkeren Klassen der Berechenbarkeitstheorie oder auch in Teilen der Mengentheorie. Was die hier behandelten Themen und Klassen zu theoretischer Informatik macht ist, dass sie, mehr oder weniger direkt, mit praktischer **Berechnung** zu tun haben. So werden reguläre Ausdrücke für das Durchsuchen von Datenbanken benutzt, die Klasse **P** ist ein gutes theoretisches Modell für in der Praxis effizient lösbare Probleme, und die Klasse der berechenbaren Funktionen stellt die Grenze praktischer Programmiersprachen dar.

Ein Aspekt der für diese, mit praktischer Berechnung in Verbindung stehenden, Klassen eine wichtige Rolle spielt ist der Unterschied zwischen **Determinismus und Nichtdeterminismus**. Wir haben bereits bei den endlichen Automaten gesehen dass DFAs und NFAs die selbe Ausdrucksstärke habe, allerdings nicht die selbe Größe. Für das **P** vs. **NP** Problem ist dieser Unterschied zentral und die Absenz starker Resultate über dieses Problem zeigt wie schlecht wir ihn bisher verstehen.

Literaturverzeichnis

[1] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[2] Piergiorgio Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Co., 1989.

[3] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[4] Michael Sipser. *Introduction to the Theory of Computation.* Cengage Learning, 3rd edition, 2012.